

2611
p 173

FINAL REPORT

DEVELOPMENT OF A NEW GENERATION SOLID ROCKET MOTOR
IGNITION COMPUTER CODE

Winfred A. Foster, Jr.
Associate Professor

Rhonald M. Jenkins
Assistant Professor

Alessandro Ciucci
Graduate Research Assistant

Shelby D. Johnson
Graduate Research Assistant

Prepared under

NASA Grant No. NAG8-683

between

George C. Marshall Space Flight Center
National Aeronautics and Space Administration

Auburn University
Engineering Experiment Station
Auburn University, Alabama 36849

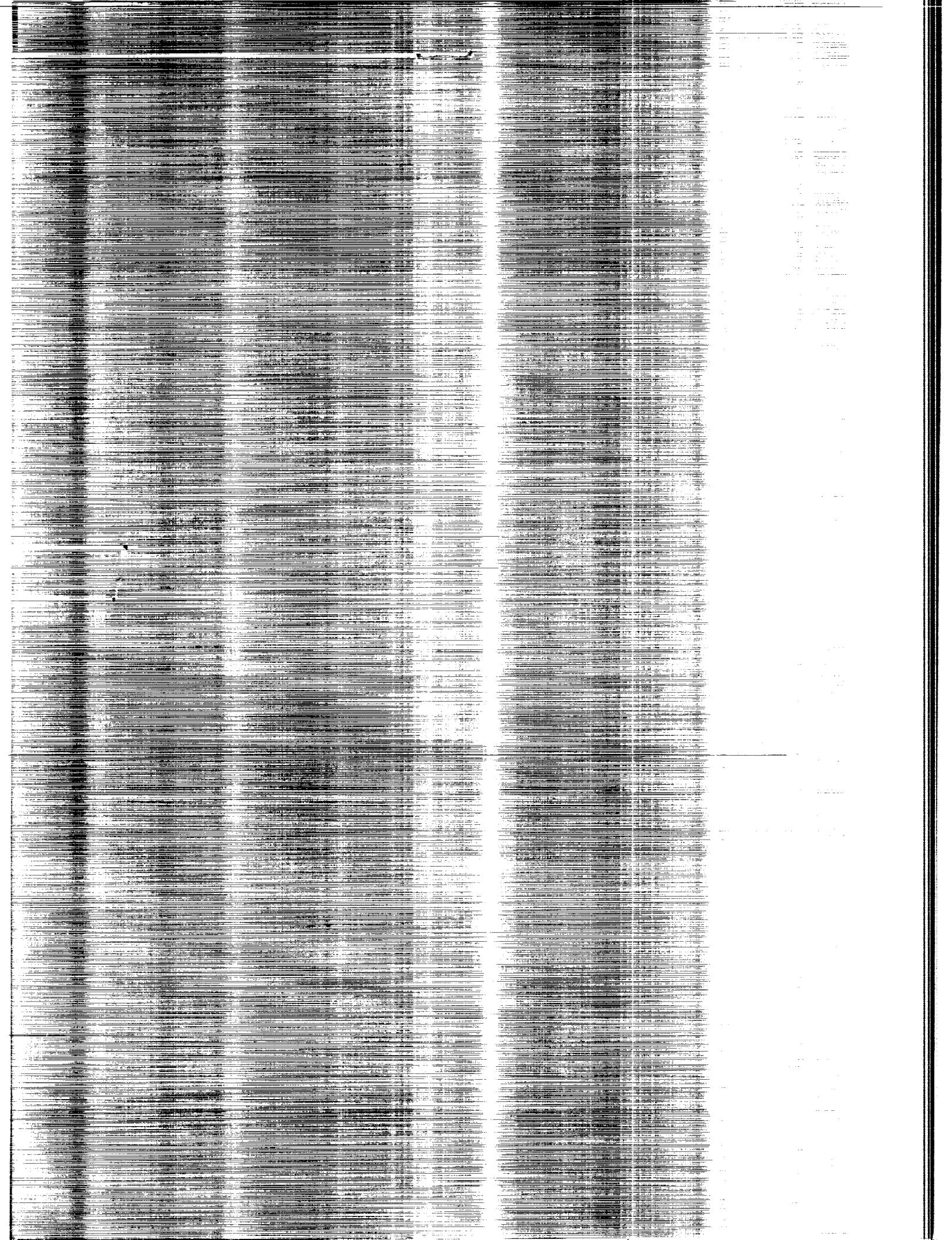
February 10, 1994

(NASA-CR-195734) DEVELOPMENT OF A
NEW GENERATION SOLID ROCKET MOTOR
IGNITION COMPUTER CODE Final Report
(Auburn Univ.) 173 p

N94-28800

Unclass

G3/20 0002611



ABSTRACT

This report presents the results of experimental and numerical investigations of the flow field in the head-end star grain slots of the Space Shuttle Solid Rocket Motor. This work provided the basis for the development of an improved solid rocket motor ignition transient code which is also described in this report. The correlation between the experimental and numerical results is excellent and provides a firm basis for the development of a fully three-dimensional solid rocket motor ignition transient computer code.

ACKNOWLEDGEMENTS

The authors express appreciation to personnel of the George C. Marshall Space Flight Center (MSFC) for their support and assistance in this project. In particular, Mr. John E. Hengel, whose overall support of this project was an essential factor in its overall success and to Mr. Andrew W. Smith for his technical assistance. The authors also wish to thank to Mr. Billy H. Holbrook for his work in constructing the experimental models.

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	vii
NOMENCLATURE.....	viii
I. INTRODUCTION	I-1
II. EXPERIMENTAL ANALYSIS	II-1
Experimental Apparatus.....	II-2
Test Facility.....	II-11
Test Plan.....	II-11
Oil Smear Results.....	II-12
Results From Static Pressure Measurements.....	II-13
Results From Heat Transfer Measurements.....	II-13
References.....	II-29
III. THEORETICAL/NUMERICAL ANALYSIS.....	III-1
Introduction.....	III-1
Conservation Equations.....	III-2
Turbulence Model.....	III-6
Heat Transfer Relations.....	III-8
Numerical Technique.....	III-10
Initial and Boundary Conditions.....	III-11
Sample Results.....	III-13
References.....	III-23
IV. CAVENY PROGRAM MODIFICATIONS.....	IV-1
References.....	IV-7
V. CAVENY PROGRAM INTERFACE.....	V-1
VI. XPLOT PROGRAM DESCRIPTION.....	VI-1
VII. XPLOT USER INSTRUCTIONS.....	VII-1
APPENDIX A Sample Caveny Input File.....	A-1
APPENDIX B Caveny Program Makefile.....	B-1
APPENDIX C xplot Source Program.....	C-1

LIST OF FIGURES

<u>FIG.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
II-1	Schematic of star slot.....	II-4
II-2	Cross-section of star slot region.....	II-4
II-3	Schematic of head-end section.....	II-8
II-4	Schematic of single-port and multi-port igniters..	II-8
II-5	Head-end section model.....	II-9
II-6	Igniters models.....	II-9
II-7	Location of pressure taps and calorimeters.....	II-10
II-8	Detail of calorimeter installation.....	II-10
II-9	Igniter 1 (100 psi).....	II-14
II-10	Igniter 2 (100 psi).....	II-14
II-11	Igniter 3 (100 psi).....	II-14
II-12	Igniter 1 (500 psi).....	II-15
II-13	Igniter 2 (500 psi).....	II-15
II-14	Igniter 3 (500 psi).....	II-15
II-15	Igniter 1 (1000 psi).....	II-16
II-16	Igniter 2 (1000 psi).....	II-16
II-17	Igniter 3 (1000 psi).....	II-16
II-18	Igniter 1 (1500 psi).....	II-17
II-19	Igniter 2 (1500 psi).....	II-17
II-20	Igniter 3 (1500 psi).....	II-17
II-21	Igniter 1 (1800 psi).....	II-18
II-22	Igniter 2 (1800 psi).....	II-18
II-23	Igniter 3 (1800 psi).....	II-19
II-24	Igniter 1 (100 psi).....	II-19
II-25	Igniter 2 (100 psi).....	II-19
II-26	Igniter 3 (100 psi).....	II-19
II-27	Igniter 1 (500 psi).....	II-20
II-28	Igniter 2 (500 psi).....	II-20
II-29	Igniter 3 (500 psi).....	II-20
II-30	Igniter 1 (1000 psi).....	II-21
II-31	Igniter 2 (1000 psi).....	II-21
II-32	Igniter 3 (1000 psi).....	II-21
II-33	Igniter 1 (1500 psi).....	II-22
II-34	Igniter 2 (1500 psi).....	II-22
II-35	Igniter 3 (1500 psi).....	II-22
II-36	Igniter 1 (1800 psi).....	II-23
II-37	Igniter 2 (1800 psi).....	II-23
II-38	Igniter 3 (1800 psi).....	II-23
II-39	Igniter 1 (100 psi).....	II-24
II-40	Igniter 2 (100 psi).....	II-24
II-41	Igniter 3 (100 psi).....	II-24
II-42	Igniter 1 (500 psi).....	II-25
II-43	Igniter 2 (500 psi).....	II-25
II-44	Igniter 3 (500 psi).....	II-25
II-45	Igniter 1 (1000 psi).....	II-26
II-46	Igniter 2 (1000 psi).....	II-26
II-47	Igniter 3 (1000 psi).....	II-26

<u>FIG.</u>	<u>TITLE</u>	<u>PAGE NO.</u>
II-48	Igniter 1 (1500 psi).....	II-27
II-49	Igniter 2 (1500 psi).....	II-27
II-50	Igniter 3 (1500 psi).....	II-27
II-51	Igniter 1 (1800 psi).....	II-28
II-52	Igniter 2 (1800 psi).....	II-28
II-53	Igniter 3 (1800 psi).....	II-28
III-1(a)	Space shuttle SRM.....	III-3
III-1(b)	SRM star grain cross section.....	III-3
III-2	SRM star grain calculation domain.....	III-12
III-3	Computational grid.....	III-12
III-4	Space Shuttle SRM igniter flow vs. time trace.....	III-12
III-5	Igniter flow-field comparison (cold flow).....	III-15
III-6	Single port igniter cold-flow, $P_{igniter} = 6.8$ atm.....	III-16
III-7	45 degree igniter cold-flow, $P_{igniter} = 6.8$ atm.....	III-17
III-8	Single port igniter cold-flow, $P_{igniter} = 34$ atm.....	III-18
III-9	45 degree igniter cold-flow, $P_{igniter} = 34$ atm.....	III-19
III-10	Typical slot area burn sequence (predicted).....	III-20
III-11	Calculated Nu Contours (cold flow) single port igniter.....	III-21
III-12	(Early) Ignition transient head-end pressure rise.....	III-22
VI-1	Two-dimensional data organization.....	VI-4
VI-2	One-dimensional data organization.....	VI-5
VI-3	X toolkit application design model.....	VI-6
VI-4	Plot layout design.....	VI-8
VI-5	Example dialog diagram.....	VI-12
VII-1	xplot top level dialog.....	VII-1
VII-2	xplot input file editor dialog.....	VII-2
VII-3	xplot execution setup dialog.....	VII-4
VII-4	Plot manager dialog.....	VII-6
VII-5	xplot format dialog.....	VII-7
VII-6	xplot data attributes dialog.....	VII-10
VII-7	xplot file editor dialog.....	VII-11
VII-8	xplot view dataset dialog.....	VII-12
VII-9	Pressure versus location at time=30 ms.....	VII-17
VII-10	Pressure versus location at time=60 ms.....	VII-17
VII-11	Pressure versus location at time=90 ms.....	VII-18
VII-12	Pressure versus location at time=120 ms.....	VII-18
VII-13	Pressure versus time at distance=3 cm.....	VII-20
VII-14	Pressure versus time at distance=1330 cm.....	VII-20
VII-15	Mach number & burn rate versus distance at time=90 ms.....	VII-21
VII-16	Mach number & burn rate versus distance at time=180 ms.....	VII-22
VII-17	Reynolds number at exit versus time.....	VII-23
VII-18	Motor thrust versus time.....	VII-23

LIST OF FIGURES

<u>Table</u>	<u>Title</u>	<u>PAGE NO.</u>
III-1	Test matrix.....	II-1
III-1	Constants used in burning rate law.....	III-9
III-2	Solid propellant properties.....	III-10
IV-1	Sample burning perimeter fraction table.....	IV-3
IV-2	Caveny code input tables.....	IV-3
IV-3	Circular perforated grain variables.....	IV-4
IV-4	Slotted grain variables.....	IV-4
IV-5	Star grain variables.....	IV-4
IV-6	Wagonwheel grain variables.....	IV-5
IV-7	Dogbone grain variables.....	IV-5
IV-8	Sample burning geometry table.....	IV-5
IV-9	Variables saved in caveny.plot file.....	IV-6
VI-1	Widgets used in xplot.....	VI-11
VI-2	xplot source files.....	VI-14
VI-3	xplot header files.....	VI-14
VII-1	Input file editor dialog commands.....	VII-4
VII-2	Execution dialog commands.....	VII-5
VII-3	Plot manager dialog commands.....	VII-6
VII-4	Format dialog commands.....	VII-9
VII-5	File editor dialog commands.....	VII-12
VII-7	Command-line interface verbs.....	VII-14
VII-8	Xplot.clr file format.....	VII-15
VII-9	Xplot.var file format.....	VII-15
VII-10	Sample Xplot.def file.....	VII-16

NOMENCLATURE

b	= star slot width
C_1, C_2	= empirical constants appearing in turbulence model
c_p	= specific heat
e	= gas internal energy
e_{gp}	= internal energy of the gaseous burning propellant
h_c	= convective heat transfer coefficient
k	= kinetic energy of turbulent velocity fluctuations
K	= thermal conductivity
M	= Mach number
Nu	= Nusselt number
p	= pressure
P_r	= Prandtl number
r	= propellant burn rate
R_e	= Reynolds number
S	= source term(s) in governing conservation equations
t	= time
T	= temperature
u, v, w	= velocity components
V	= $(u^2 + v^2)^{1/2}$
x, y, z	= spatial coordinates
α	= thermal diffusivity
γ	= ratio of specific heats
ϵ	= rate of dissipation of turbulent kinetic energy
$\sigma_k, \sigma_\epsilon$	= constants appearing in turbulence model
σ_p	= constant appearing in propellant burn rate equation

subscripts

c	= mass
e	= energy
i	= initial value
l	= laminar
p	= propellant
ref, R	= reference condition
star tip	= propellant grain star tip
t	= turbulent
wa	= adiabatic wall
xm, ym	= x, y components of linear momentum

superscript

*	= dimensionless quantity
---	--------------------------



I. INTRODUCTION

This report describes the results obtained from experimental and numerical analyses conducted for the purpose of developing an improved solid rocket motor ignition transient code. The specific objective of this work was to improve ability to predict the influence of the star grain on the ignition transient; in particular, the calculation of the flame spreading rate on the propellant surfaces inside the star slot. This report is divided into three main subject areas: 1) experimental analysis, 2) numerical analysis and 3) computer code modification, development and documentation.

Section II presents the results of a series of experiments conducted at NASA's George C. Marshall Space Flight Center (MSFC). These experiments utilized a model designed and constructed in the Aerospace Engineering Department at Auburn University. This model is a one-tenth scale simulation of the Space Shuttle Solid Rocket Motor (SRM) head-end section. The model was tested in the special test section of the 14" x 14" trisonic wind tunnel at MSFC. The tests were cold-flow, using air, simulations of the internal flow through the igniter nozzle and the head-end section of the SRM. The air was supplied through a special high pressure line connected to the wind tunnel. There was no external flow around the model. The tests were designed to provide both qualitative and quantitative data on the interaction between the igniter plume and the star slots and the flow field within the star slots. Qualitative measurements were made using oil smears, Schlieren photography and by seeding the flow field with aluminum particles which were illuminated with a laser system and recorded on video tape. Quantitative measurements were made of the pressure distribution within a slot and of the heat transfer rates to the wall of a slot. The correlation of the data between the various experiments performed is excellent.

Section III provides the theoretical basis for the Computational Fluid Dynamic (CFD) model which was developed to analyze the flow field in a star slot and the results obtained from the CFD analysis. The CFD model was verified using the experimental data obtained from the cold flow tests described above. The primary objective of the CFD model was to provide data regarding the spread of the combustion process throughout the star slots. Based on these calculations a burning surface area versus time model is obtained for use in the ignition transient performance model of an SRM. The results of the analysis are compared to existing motor data and are shown to be in good agreement.

Sections IV-VII describe the modifications, interface, program description, and instructions for an improved ignition

transient computer code. The modifications and interface are to an existing one-dimensional ignition transient computer code. The code was chosen because it is well documented and provides an efficient means of implementing the modifications to the burning surface versus time calculations as determined from the analyses described above. The ability to account for the flame spreading in the star slot of the head-end section represents a significant step forward in the ability to accurately model the early portion of an SRM's ignition transient. The last two sections describe the development of a pre- and post-processor for an ignition transient code. The code presented here uses a modified one-dimensional ignition transient code modified to account for the flame spreading in the star slots to solve for the ignition transient performance of an SRM. However, it would be relatively straightforward to modify the current version of the code to use a more sophisticated ignition transient model as a solver. The computer code is currently implemented on a SUN workstation using X-Windows. Instructions necessary for using the program in an X-Windows environment are included in Section VII. A sample input file and the code for the pre- and post-processor are included in the Appendices.

The work described in this report provides an excellent base for the development of a fully three-dimensional ignition transient performance prediction code using CFD techniques. The experimental database which has been generated from this work provides significant new insight into flow field phenomena occurring in the star slots of SRM's which have head-end star grains and head-end igniters.

II. EXPERIMENTAL ANALYSIS

One of the primary limitations of existing ignition transient prediction computer codes is an overestimation of the ignition delay and departure of the predicted pressure-time history curve from measured data in the early part of the transient. One of the reasons for this is the lack of data on the flow field in star grain sections of solid rocket motors. The experiments described in this section focused on the flow field in the star grain section of the Space Shuttle solid rocket motor (RSRM). The purpose of the experiments was to obtain a credible data base for the flow field patterns and heat transfer rates within the star slots. A one-tenth scale model of the Space Shuttle RSRM head-end star grain was designed and constructed for the experimental study. The availability of experimental flow field data during the ignition transient in solid rocket motors is very scarce. Conover¹ conducted cold flow tests using a one-tenth scale model of the Space Shuttle solid rocket motor's head-end star grain section. Conover's tests used both a single port igniter, such as found on the RSRM and a three port igniter. The series of tests included Schlieren photographs of the igniter mounted in a plenum, oil smear data, pressure data and heat transfer coefficient measurements. Fifteen pressure ports were fitted in one side plate of a slot, while fifteen calorimeters to measure heat transfer coefficients were located in another slot. The data were taken at pressure levels from 100 to 1500 psi in 100 psi increments. The actual Space Shuttle igniter operates at approximately 2000 psi. Limits on the test facility prevented taking data at higher igniter chamber pressures. The static pressure data obtained provide some qualitative trends, but there was considerable scatter in the data when the igniter chamber pressure exceeded 1100 psi. This was probably due to the fact, as Conover states, that "above this pressure the side plates used to form the star grain slots were deflected to produce a one-sixteenth-inch gap where the plates come together at the star points," thus causing some leakage. There were also some apparent inconsistencies in the temperature data due to model warm-up during the test. However, the oil smear data provide a good indication of the recirculating flow pattern inside a slot. Even though Conover's experimental data provide useful information on the flow field inside the star slot, they should be considered preliminary in nature, and a starting point for a more in-depth investigation.

The present work describes a series of tests directed toward the collection of qualitative and quantitative data documenting the main characteristics of the flow in the head-end section of a solid rocket motor. In particular, the following objectives were to be accomplished with the test program.

- 1) Obtain information on the igniter plume structure and shape and its interaction with the star grain geometry.
- 2) Determine the region of the igniter plume impingement on the side walls of the slots of the RSRM star grain section model.
- 3) Determine the flow field characteristics of the subsonic, recirculating flow within the slot.
- 4) Measure the heat transfer coefficients at several locations inside the slot.

The above objectives were to be accomplished using the following techniques:

- 1) Flow visualization.
- 2) Oil smears
- 3) Static pressure measurements
- 4) Heat transfer coefficient measurements
- 5) Velocity measurements

Experimental Apparatus

The test article used in this investigation was a one-tenth scale, cold flow model based on the geometry of the Space Shuttle RSRM head-end section. The test article had four slots, as opposed to eleven slots in the actual Space Shuttle motor. A single port igniter model and two four-port igniter models were used in the tests.

The scale factor of 1:10 was derived from an analysis which matches the Reynolds number between the model flow and the full scale flow in the star grain section. Besides geometrical similitude, the primary scaling parameter to ensure proper similarity between the cold flow model and the real, full scale motor is the Reynolds number. Compressibility effects are important only in the igniter plume region, since the flow inside a star slot is essentially subsonic. However, the igniter mass flow rate is an important parameter, since it is thought to be responsible for entrainment of the flow, which determines the recirculation pattern. The value of the Reynolds number determines the nature of the viscous effects. The viscous effects in turn are related to the local convective heat transfer coefficient and therefore to the amount of heat transfer from the hot gases to the solid propellant. An exact match of Reynolds number between the model and full scale flow field is not necessary for similarity. Instead, generally good agreement between Reynolds numbers is a sufficient condition for the general studies of these flow fields.

as has been extensively documented in the literature.²⁻⁵

Because of the complicated nature of the flow in the head-end section, it is difficult to define an overall Reynolds number for the entire slot. However, it seems appropriate to consider a representative Reynolds number, which can be defined at any point in the star slot, as:

$$R_e = \frac{\rho V L}{\mu} \quad \text{II-1}$$

where ρ , V and μ are the local density, velocity and viscosity, respectively, and L is a geometric reference length (for instance, the distance from a given point to the motor centerline). The product ρV can be viewed as the local mass flux. It is assumed that this local mass flux is proportional to the overall mass flux that enters a star slot, which in turn is proportional to the mass flow rate that enters the star slot divided by the open area of the slot, so that:

$$\rho V = \frac{\dot{m}_{\text{slot,act}}}{bl} \quad \text{II-2}$$

where b and l are the slot width and slot length, respectively, as shown in Figure II-1.

The mass flow rate that actually enters a given slot depends on several factors, such as igniter mass flow rate, \dot{m}_{ig} , geometric dimensions of the slot and motor, and igniter plume shape. The functional dependency between these parameters is usually not known, but can be generally expressed as:

$$\dot{m}_{\text{slot,act}} = \dot{m}_{\text{slot,avail}} f(\text{plume shape}) \quad \text{II-3}$$

To simplify the analysis, it can be assumed that three-dimensional effects are negligible and the mass flow available to the slot, $\dot{m}_{\text{slot,avail}}$, is the portion of igniter mass flow encompassed in the circular sector facing the same slot (see Fig. II-2), or

$$\dot{m}_{\text{slot,avail}} \approx \dot{m}_{ig} \frac{\theta}{2\pi} \approx \dot{m}_{ig} \frac{b}{2\pi r} \quad \text{II-4}$$

The factor $b/(2\pi r)$ can be considered as the fraction of the port perimeter occupied by a slot.

Using Eqs. (II-2), (II-3) and (II-4), the Reynolds number can be expressed as:

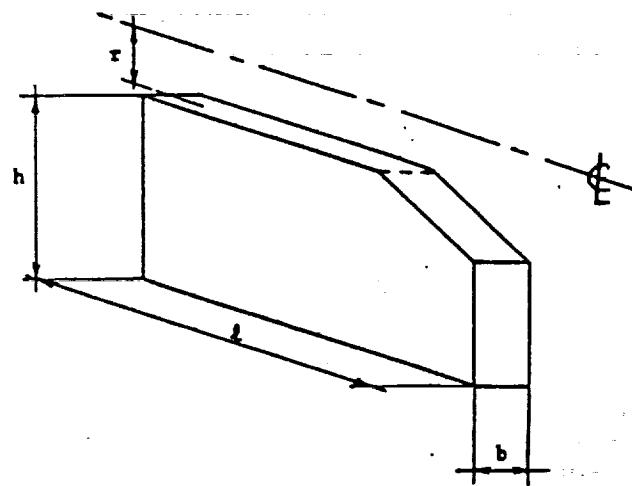


Fig. II-1 Schematic of star slot

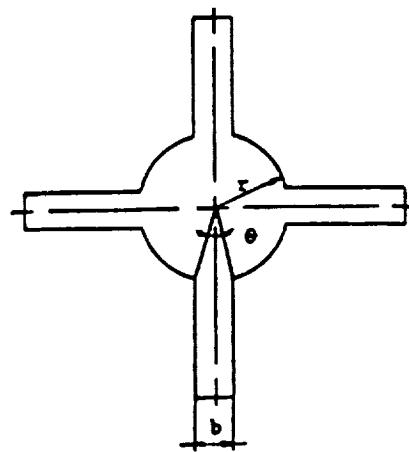


Fig. II-2 Cross-section of star slot region

$$R_e \approx \frac{\dot{m}_{ig}}{2\pi r} \frac{L}{\mu} f(\text{plumeshape})$$

II-5

Note that the Reynolds number, and thus the flow pattern inside the slot, is independent of the number of slots.

For proper scaling, let $R_{e1} = R_{e2}$, where the subscript 1 refers to the actual motor and 2 refers to the model. Then the following assumptions are made:

- 1) All dimensions are exactly scaled.
- 2) The igniter plume shape is scaled for 1 and 2.

Note that the plume shape is determined mainly by the igniter nozzle area ratio, igniter stagnation pressure, P_0 , and back pressure. When all dimensions are scaled, the area ratio is the same for the real srm and the scaled model. The back pressure is also the same, and equal to the ambient value. This latter condition is true at least until the point in time when a first ignition occurs. This suggests that the igniter total pressure ideally should be the same, that is 2000 psi.

Under the assumptions above, $f(\text{plume shape})$ drops out when equating the Reynolds numbers, and the following relationship is obtained:

$$\frac{r_2}{r_1} \frac{l_2}{l_1} \frac{L_1}{L_2} = \frac{\dot{m}_{ig2}}{\dot{m}_{ig1}} \frac{\mu_1}{\mu_2} \quad \text{II-6}$$

Then, defining the scaling factor as S , so that

$$S = \frac{L_2}{L_1} = \frac{l_2}{l_1} = \frac{r_2}{r_1} \quad \text{II-7}$$

Equation (II-6) is reduced to:

$$S = \frac{\dot{m}_{ig2}}{\dot{m}_{ig1}} \frac{\mu_1}{\mu_2} \quad \text{II-8}$$

At this point, an appropriate viscosity-temperature relationship must be chosen. Following Caveny⁵, an empirical curve fit for

typical combustion gases is such that viscosity, μ , is proportional to the gas temperature, T , as

$$\mu = T^{0.6}$$

II-9

so that Eq. (II-8) becomes:

$$S = \frac{\dot{m}_{ig2}}{\dot{m}_{ig1}} \left(\frac{T_1}{T_2} \right)^{0.6}$$

II-10

Since \dot{m}_{ig1} , T_1 and T_2 are known values, it appears from Eq. (II-10) that the scale factor, S , can be arbitrarily chosen simply by varying the igniter model mass flow rate, \dot{m}_{ig2} . However, the igniter throat area is dependent on the scale factor to be determined. Furthermore, the igniter chamber pressure should be close to 2000 psi because of the assumption that the plume shape does not change from the real motor to the scaled model. This is more evident if Eq. (II-10) is rewritten in terms of stagnation pressures and temperatures, as:

$$S = \frac{P_{01}}{P_{02}} \sqrt{\frac{R_2}{R_1} \frac{T_{02}}{T_{01}}} \left(\frac{T_2}{T_1} \right)^{0.6} \sqrt{\frac{\gamma_1}{\gamma_2} \frac{\left(\frac{2}{\gamma_1+1} \right)^{\frac{\gamma_1+1}{2(\gamma_1-1)}}}{\left(\frac{2}{\gamma_2+1} \right)^{\frac{\gamma_2+1}{2(\gamma_2-1)}}}}$$

II-11

Note that the term A_2^*/A_1^* is, itself, S^2 .

For the cold-flow tests, the values of R_2 and γ_2 for air are chosen for the model, and typical values of the temperatures are $T_{02} = 310^\circ\text{K}$, $T_2 = 298^\circ\text{K}$. Since $P_{01} = P_{02} = 2000$ psi, the pressure term does not contribute to the value of S . Representative values of the variables for the flow in the head-end section of the actual SRM are taken from Ref. 5.

Substituting in Eq. (II-11), a value equal to 0.098 is obtained for S , which is very close to the chosen scale factor 1:10. Fortunately, a one-tenth scale also defines a model size which is close to the largest size that would fit in the dimensions of the test section of the 14x14-inch tunnel employed for the tests.

The number of slots was determined based on different criteria. According to the previous analysis, the Reynolds number is independent of the number of slots. Therefore, one can ideally choose any convenient number. Even though the actual SRM head-end

section has eleven slots, only four slots are used in the scaled model. This represents a tradeoff between the desirability of flow visualization in one (transparent) star slot and the requirements for all other test instrumentation.

The entire star grain section model, as well as the three igniter models, were fabricated from aluminum, except for the transparent slot which consists of two plexiglas plates. The total length of the model is 19.72 inches; the largest diameter is 16 inches. Figures II-3 and II-4 show a schematic representation of the entire model and of the igniters, respectively.

Each star slot is formed from two plates separated by a bottom spacer of suitable thickness. An insert at the head-end simulates the actual grain surface. The circular port is formed from four contoured pieces connected to the outer surface of the slot plates. Two plates are located at the upstream and downstream end to close the slots, and provide the attachment points for all the parts. The igniter is connected to the inner surface of the head-end plate. The single port igniter has an insert into which the nozzle has been cut. The single port igniter has a throat diameter of 0.6025 inches and a conical shape with a half-angle of 27.2 degrees. The area ratio is 1.428. The four-port igniters are simply four straight holes drilled in the igniter casing, each with a 0.3012 diameter. The four-port igniters are oriented so that each of the four jets centerlines are directed into a star slot. Figure II-5 shows the whole star grain section model. The three igniters used are shown in Fig. II-6.

The model is fully instrumented for measuring the parameters of interest as defined above.

Static pressure measurements are taken inside a single star slot and along two of the four contoured sectors forming the circular port of the model. Three pressure ports are located along each of the two contoured sectors. Twenty-eight static pressure ports are provided in one wall of a star slot. The measurements are taken at three different slot depths and consist of eight, ten, and eight pressure taps, respectively, as shown in Fig. II-7. The three depths are equally spaced along the height of the slot.

In the plate forming the wall adjacent to the one containing the pressure taps, twenty-eight calorimeters are installed to obtain heat transfer coefficient data. The calorimeters are placed at the same geometric locations as the pressure ports (Fig. II-9). Each calorimeter is mounted in a plug, flush with the inner surface of the slot plate. A detail of the calorimeter installation is given in Fig. II-8. In order to get accurate measurements of the heat transfer coefficients, the calorimeters were pre-heated before each measurement was taken.

A second star is used to obtain oil smear data. A silicone-

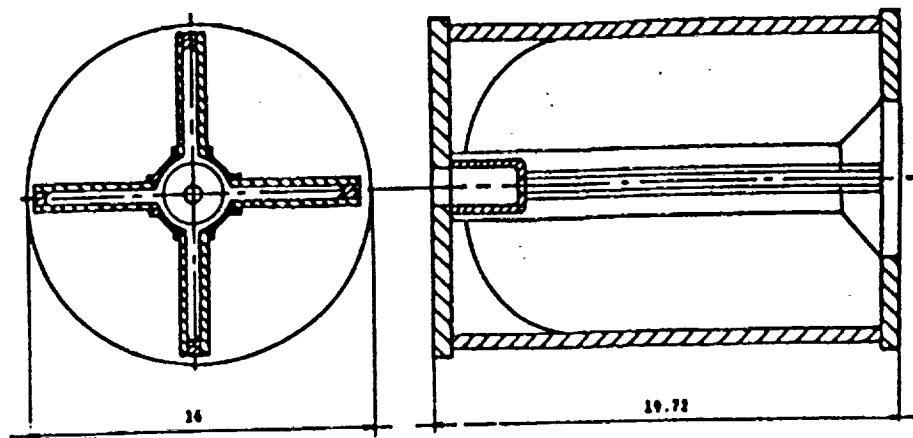


Fig. II-3 Schematic of head-end section

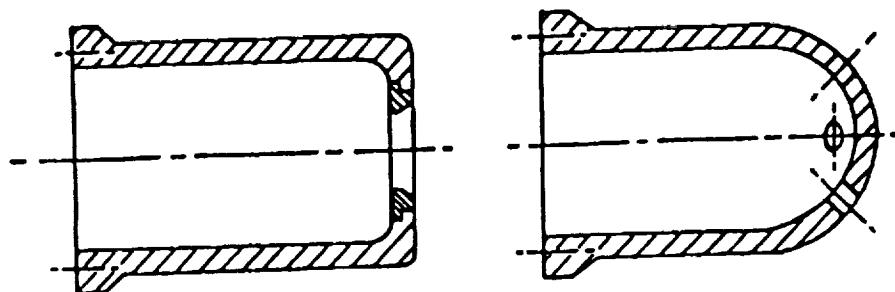


Fig. II-4 Schematic of single-port and multi-port igniters

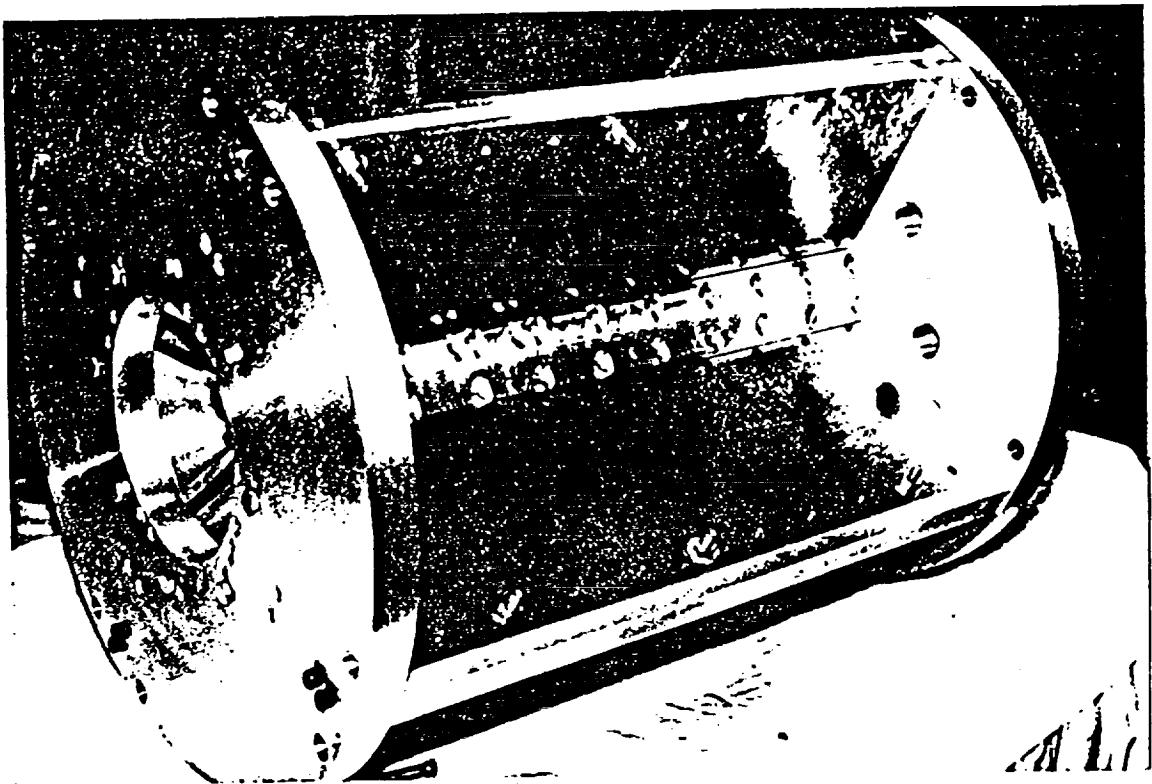


Fig. II-5 Head-end section model

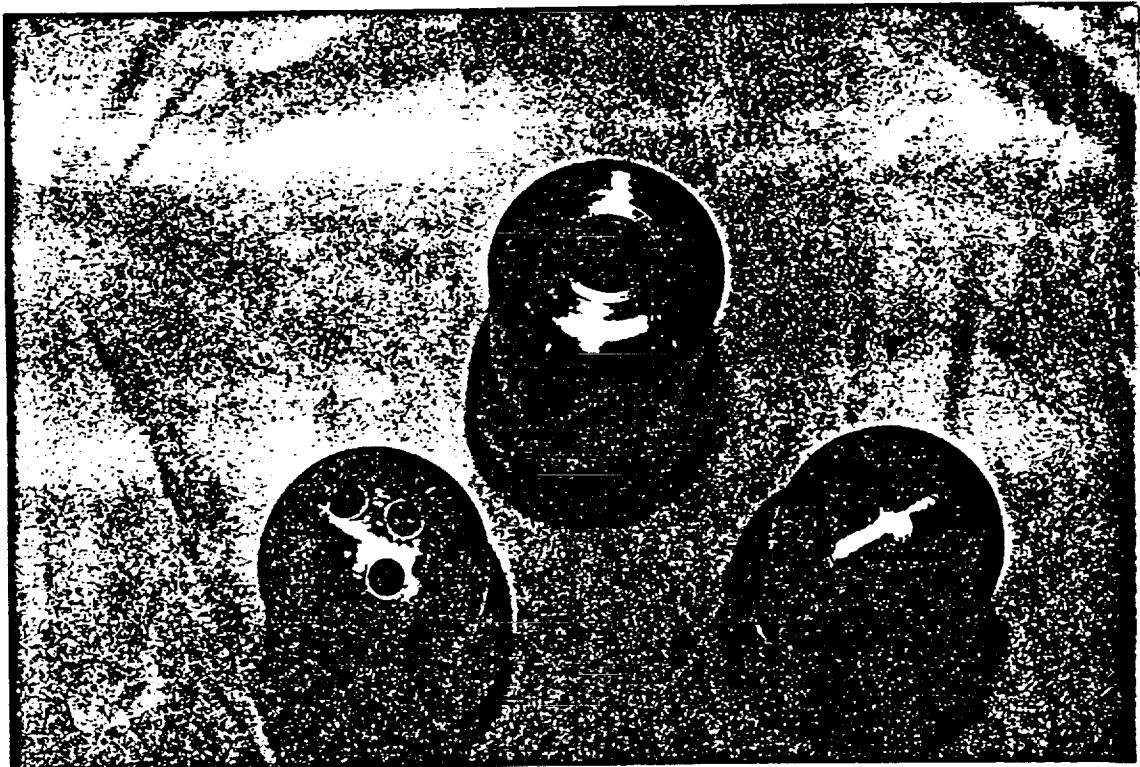


Fig. II-6 Igniter models

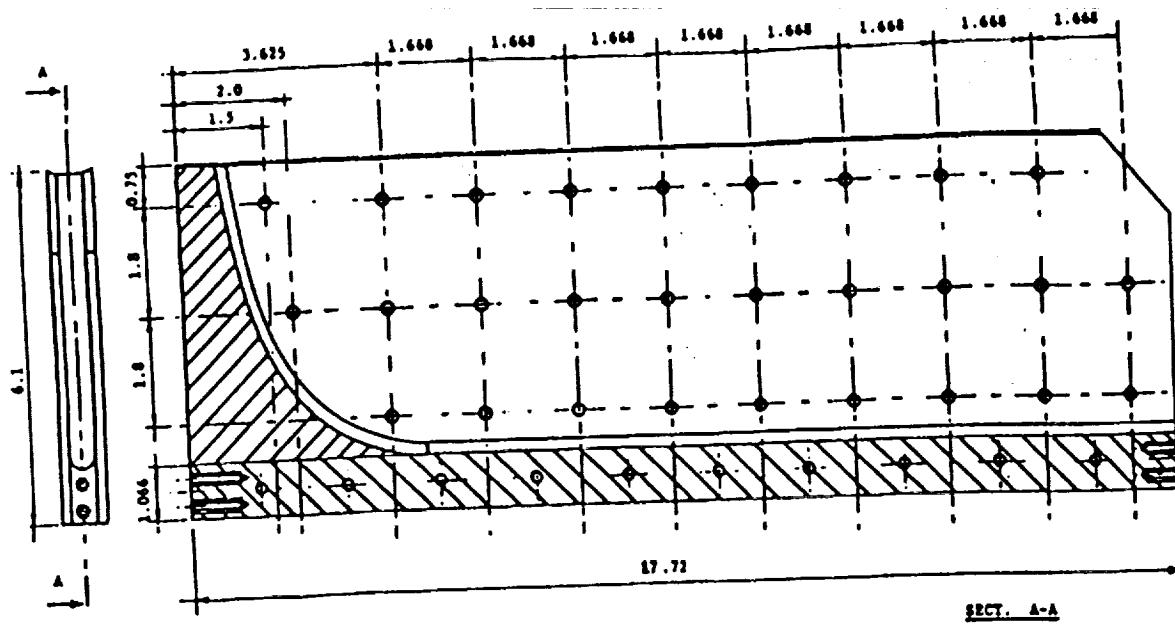


Fig. II-7 Location of pressure taps and calorimeters

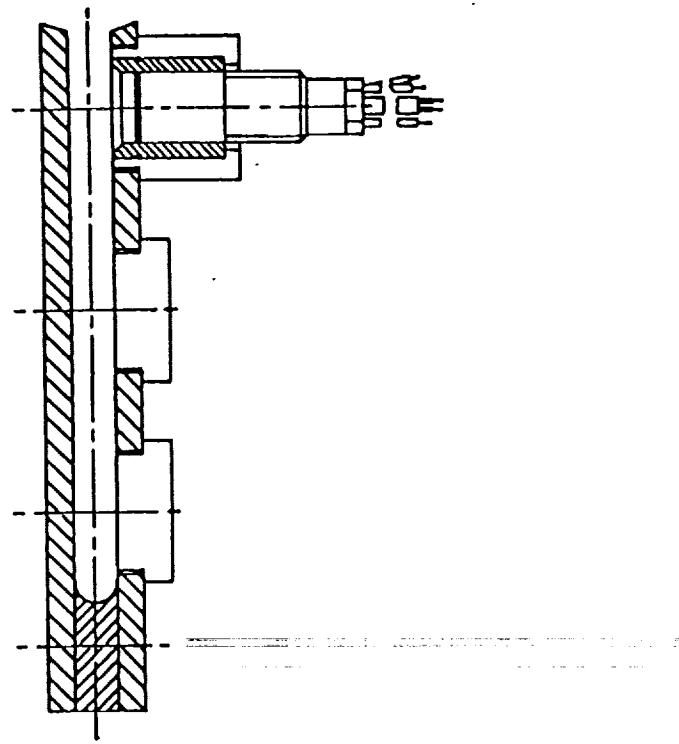


Fig. II-8 Detail of calorimeter installation

based oil was used to apply a matrix of oil drops to the finished surface of one of the plates in the slot. After each run, well defined marks or smears indicated the local direction of the flow and gave a good overall picture of the flow field.

The transparent slot previously mentioned was used for real time flow visualization. A laser sheet was projected from the aft-end of the model and illuminated most of the transparent slot. Aluminum particles mixed with pure alcohol were injected into the slot and the aluminum particles were illuminated by the laser sheet. The movement of the particles was clearly visible in the transparent slot and provided an excellent qualitative measurement of the behavior of the flow field in the slot. The flow visualization obtained using the aluminum particles gave a more detailed picture of the flow patterns in the slot than was possible with the oil smears. In addition, the real time nature of measurements provided a means for studying the dynamic characteristics of the flow field. Video tape recordings of these experiments were made to document the measurements.

The fourth slot was initially intended for making hot-wire anemometry measurements. However, because of difficulties associated with flow blockage in the small slot and difficulties in making measurements in or near the high speed (high subsonic or low supersonic) plume, this measurement was abandoned for the present investigation.

Test Facility

The cold-flow tests used the special test section in the NASA Marshall Space Flight Center 14x14-inch trisonic wind tunnel. The tunnel operated as an intermittent blow-down wind tunnel from storage pressure to atmospheric exhaust. The full Mach number capability was not needed for the test program which was carried out. Instead only a high pressure internal flow through the special test section was required. The high pressure air passed through the hollow centerbody of the tunnel, into a pipe connected to the head-end plate of the model. It was then exhausted into the special test section through the igniter models. There was no external flow around the model. A venturi was installed upstream of the model to determine the mass flow rate through the igniter. Additional information regarding the NASA/MSFC trisonic wind tunnel is given in Ref. 6.

Test Plan

The test program included two main series of tests. Table II-1 shows the test matrix which was used for each series of tests. In the first series each igniter model was placed in the test section without the star grain portion of the model. Air flow at pressures of 100, 500, 1000, 1500 and 1800 psi passed through each of the igniters used in the experiments and mass

flow rates corresponding to each pressure were recorded. A Schlieren system and video tape recorder were used during each run to examine the plume shape at various pressures. This was done to establish a reference for the plume geometry which could be compared with the plume geometry observed with the star grain in place.

The second series of tests used the entire head-end section model along with each of the three igniters. Oil smear, flow visualization, static pressure and heat transfer coefficient measurements were made at each condition shown in the test matrix of Table II-1. It should be noted that the test condition at 1800 psi approximates the design condition of 2000 psi at which similitude between the one-tenth scale model and the actual Space Shuttle RSRM flow field is achieved. The value of 1800 psi was used because it represents the upper limit on the facility at the mass flow rates necessary for the tests.

Table II-1. Test matrix

Igniter Angle (deg)	Chamber Pressure (psi)	100	500	1000	1500	1800
0		1	2	3	4	5
22.5		6	7	8	9	10
45		11	12	13	14	15

Oil Smear Test Results

Oil smears were taken for each of the fifteen test conditions shown in Table II-1. The oil smears were generated from a pattern of oil droplets placed on one side of a slot. The spacing between the droplets was approximately one-half inch. Figs. II-9 through II-23 show photographs of the oil smears generated for each of the fifteen test conditions. The oil smears provide considerable detail regarding the direction of the primary flow in the slots, the region of the igniter plume impingement, and the recirculation patterns which occur in the slot. Although qualitative in nature, this data showed good agreement with the data from the CFD analyses presented in refs. 7-9, which are summarized in Section III of this report.

Results From Static Pressure Measurements

Static pressure measurements were made at 27 locations on the surface of one of the slots. Fig. II-7 shows the location and numbering scheme used for the static pressure ports. Static pressure distributions obtained from these measurements are shown in Figs. II-24 through II-38. This data confirms the quantitative data obtained from the oil smear tests with regard to the location of the main flow paths, recirculation regions, stagnation points and "dead" regions within the slot. The data obtained agrees well with the CFD results which will be discussed in Section III, where a comparison will be given between the experimental data and the CFD results.

Results From Heat Transfer Measurements

Calorimeters were placed in a slot face adjacent to the slot face where the static pressure measurements were made. The calorimeters were located at points corresponding to the 27 locations shown in Fig. II-7. Because of the lack of a sufficient number of calorimeters to measure data at all 27 locations simultaneously, two runs were made using 15 calorimeters in each run. Three calorimeters were not moved between runs to insure that consistent data were being obtained between the two runs. The results for the measured heat transfer coefficients are shown in Figs. II-39 through II-53.

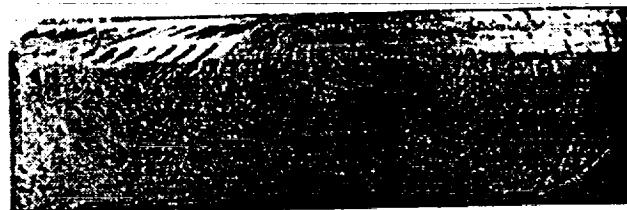


Fig. II-9 Igniter 1 (100 psi)

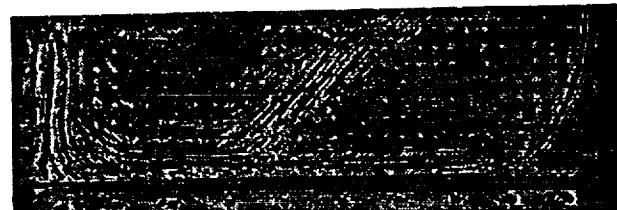


Fig. II-10 Igniter 2 (100 psi)

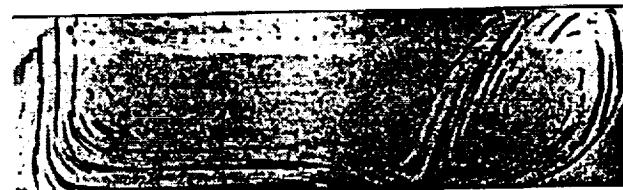


Fig. II-11 Igniter 3 (100 psi)



Fig. II-12 Igniter 1 (500 psi)

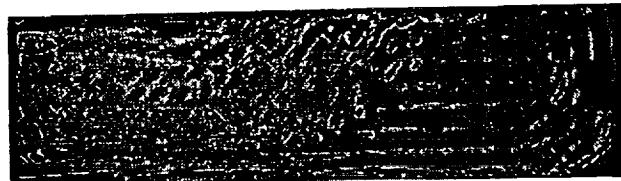


Fig. II-13 Igniter 2 (500 psi)



Fig. II-14 Igniter 3 (500 psi)



Fig. II-15 Igniter 1 (1000 psi)



Fig. II-16 Igniter 2 (1000 psi)

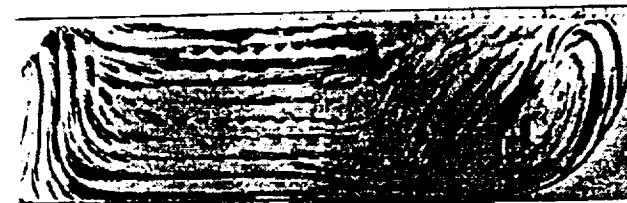


Fig. II-17 Igniter 3 (1000 psi)

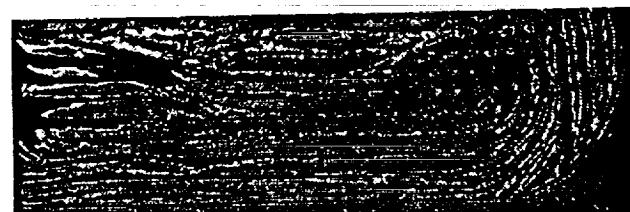


Fig. II-18 Igniter 1 (1500 psi)

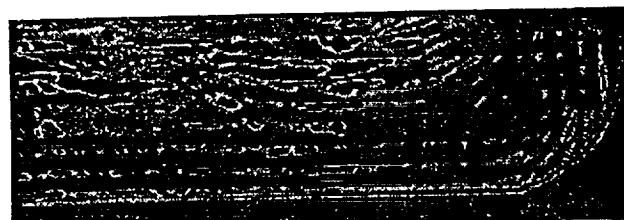


Fig. II-19 Igniter 2 (1500 psi)



Fig. II-20 Igniter 3 (1500 psi)



Fig. II-21 Igniter 1 (1800 psi)

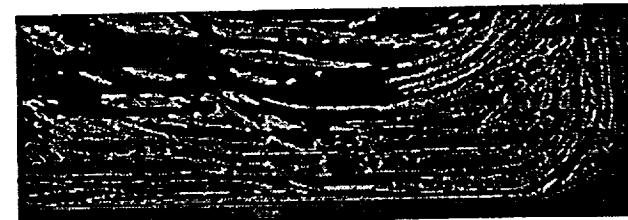


Fig. II-22 Igniter 2 (1800 psi)

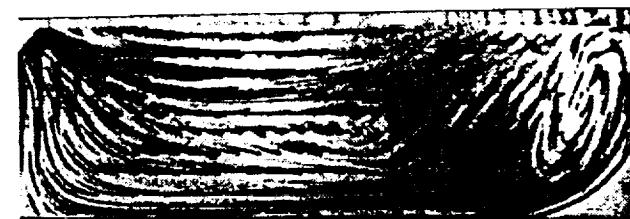


Fig. II-23 Igniter 3 (1800 psi)

Measured Pressure, Single Port Igniter

Pigniter = 100 psi

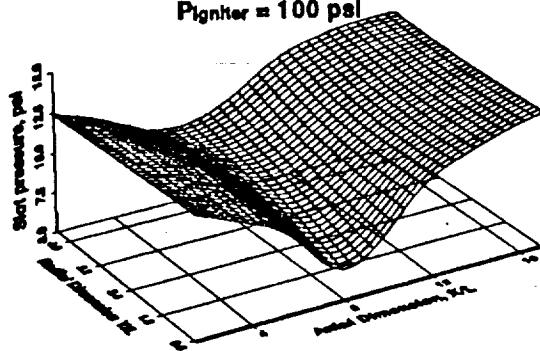


Fig. II-24 Igniter 1 (100 psi)

Measured Pressure, 22 deg Igniter

Pigniter = 100 psi

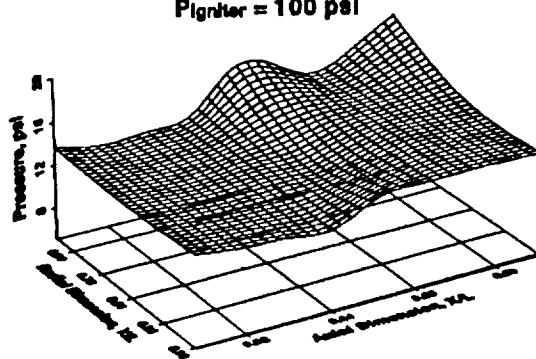


Fig. II-25 Igniter 2 (100 psi)

Measured Pressure, 45 deg Igniter

Pigniter = 100 psi

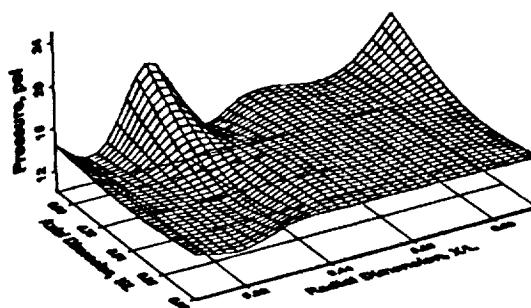


Fig. II-26 Igniter 3 (100 psi)

Measured Pressure, Single Port Igniter

P_{igniter} = 500 psi

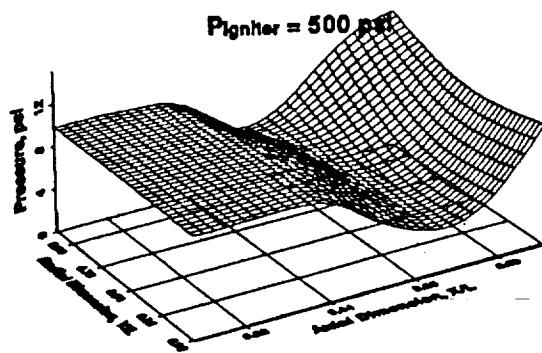


Fig. II-27 Igniter 1 (500 psi)

Measured Pressure, 22 deg Igniter

P_{igniter} = 500 psi

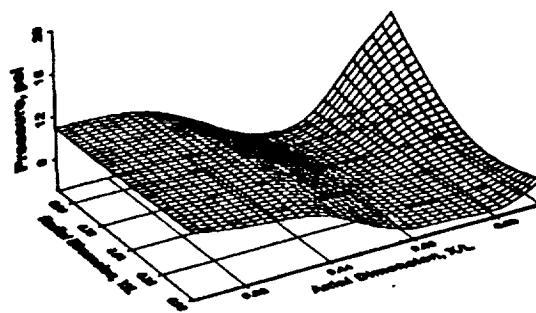


Fig. II-28 Igniter 2 (500 psi)

Measured Pressure, 45 deg Igniter

P_{igniter} = 100 psi

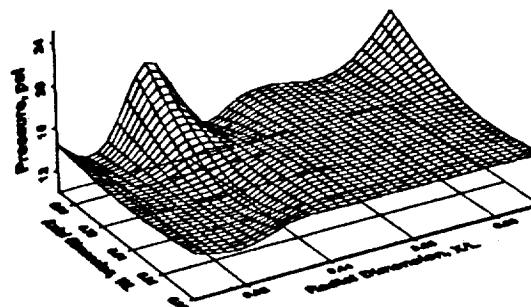


Fig. II-29 Igniter 3 (500 psi)

Measured Pressure, Single Port Igniter

P_{igniter} = 1000 psi

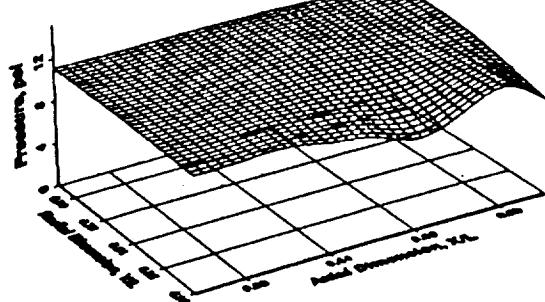


Fig. II-30 Igniter 1 (1000 psi)

Measured Pressure, 22 deg Igniter

P_{igniter} = 1000 psi

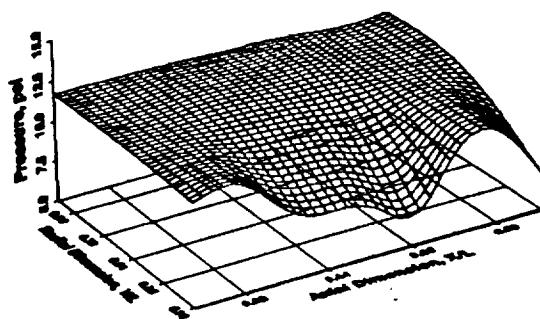


Fig. II-31 Igniter 2 (1000 psi)

Measured Pressure, 45 deg Igniter

P_{igniter} = 1000 psi

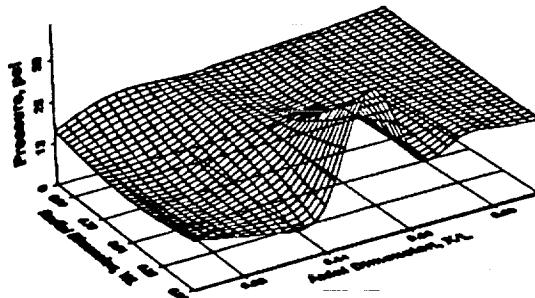


Fig. II-32 Igniter 3 (1000 psi)

Measured Pressure, Single Port Igniter

Igniter = 1500 psi

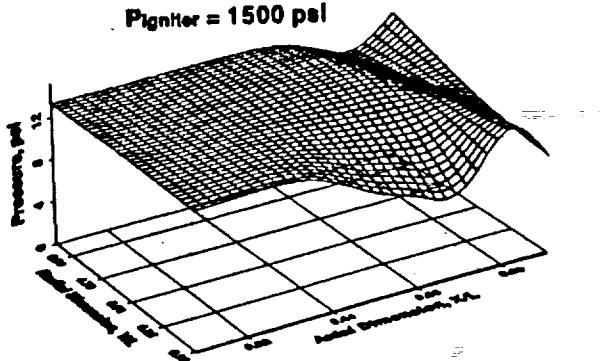


Fig. II-33 Igniter 1 (1500 psi)

Measured Pressure, 22 deg Igniter

Igniter = 1500 psi

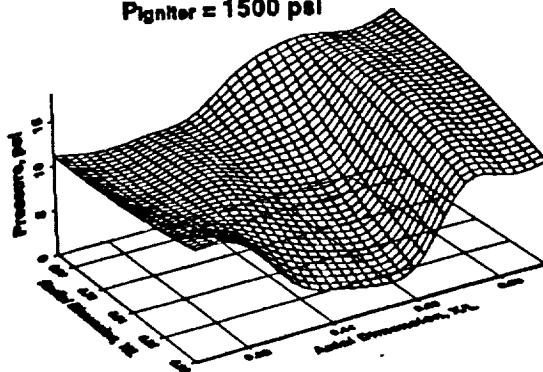


Fig. II-34 Igniter 2 (1500 psi)

Measured Pressure, 45 deg Igniter

Igniter = 1500 psi

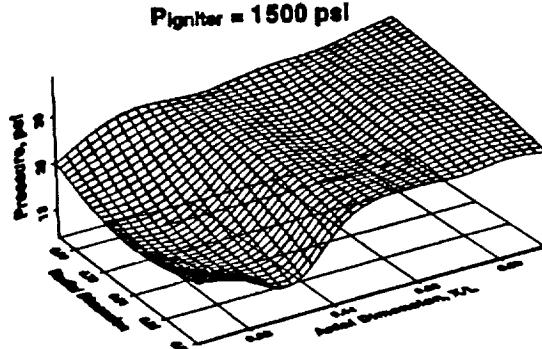


Fig. II-35 Igniter 3 (1500 psi)

Measured Pressure, Single Port Igniter

Igniter = 1800 psi

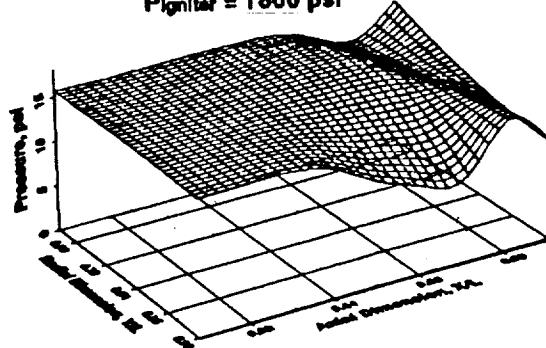


Fig. II-36 Igniter 1 (1800 psi)

Measured Pressure, 22 deg Igniter

Igniter = 1800 psi

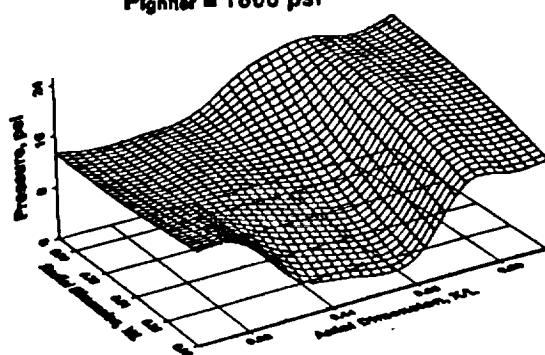


Fig. II-37 Igniter 2 (1800 psi)

Measured Pressure, 45 deg Igniter

Igniter = 1800 psi

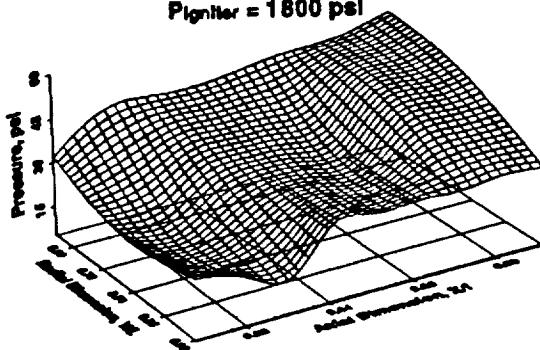


Fig. II-38 Igniter 3 (1800 psi)

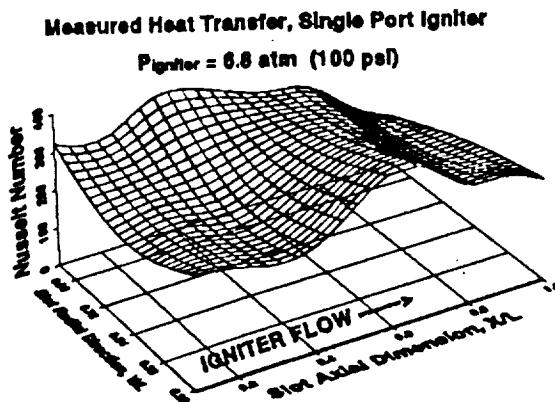


Fig. II-39 Igniter 1 (100 psi)

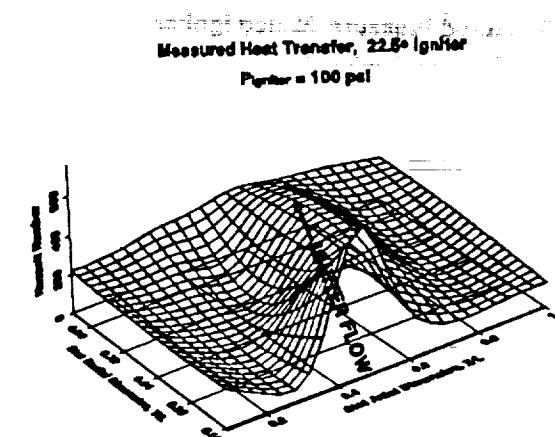


Fig. II-40 Igniter 2 (100 psi)

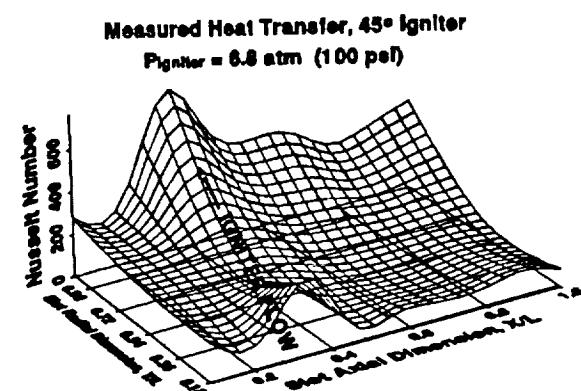


Fig. II-41 Igniter 3 (100 psi)

Measured Heat Transfer, Single Port Igniter
P_{igniter} = 500 psi

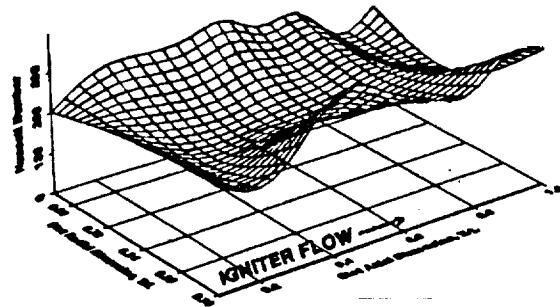


Fig. II-42 Igniter 1 (500 psi)

Measured Heat Transfer, 22.8° Igniter
P_{igniter} = 500 psi

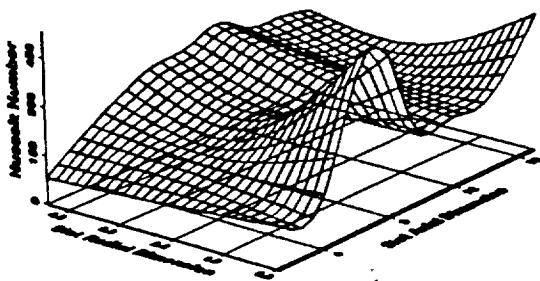


Fig. II-43 Igniter 2 (500 psi)

Measured Heat Transfer, 45° Igniter
P_{igniter} = 500 psi

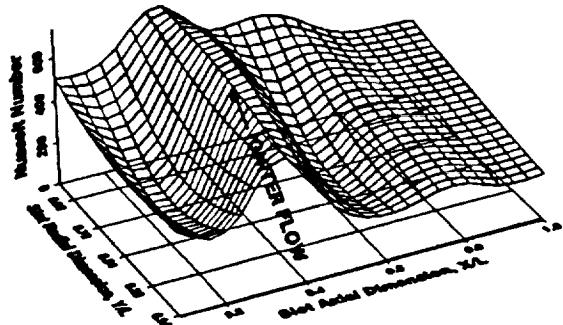


Fig. II-44 Igniter 3 (500 psi)

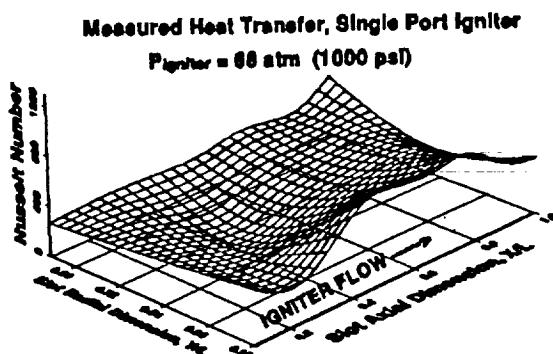


Fig. II-45 Igniter 1 (1000 psi)

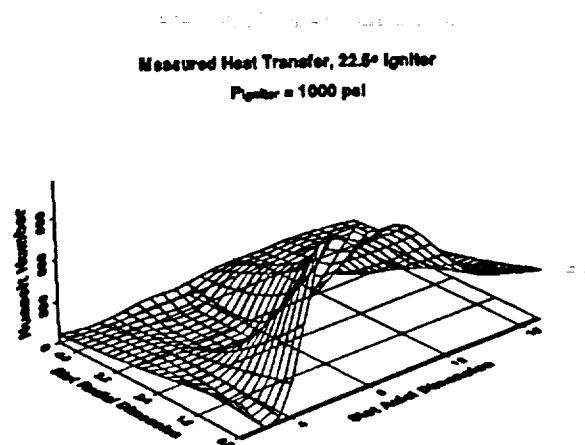


Fig. II-46 Igniter 2 (1000 psi)

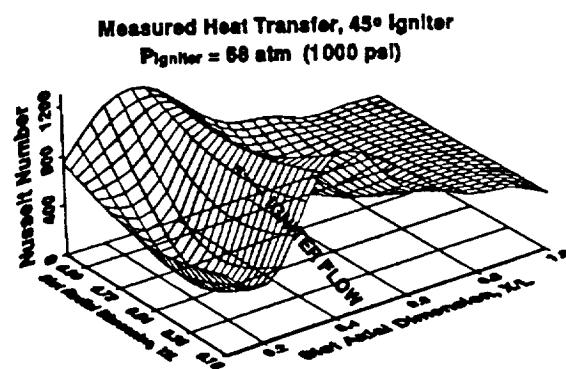


Fig. II-47 Igniter 3 (1000 psi)

Measured Heat Transfer, Single Port Igniter

$P_{igniter} = 1500$ psi

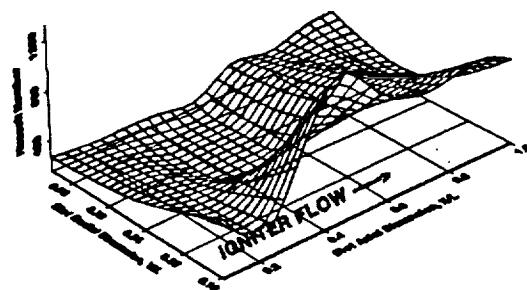


Fig. II-48 Igniter 1 (1500 psi)

Measured Heat Transfer, 22.5° Igniter

$P_{igniter} = 1500$ psi

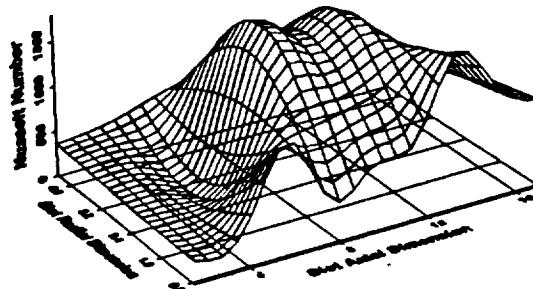


Fig. II-49 Igniter 2 (1500 psi)

Measured Heat Transfer, 45° Igniter

$P_{igniter} = 1500$ psi

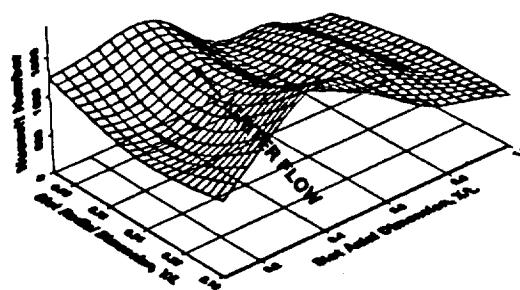


Fig. II-50 Igniter 3 (1500 psi)

Measured Heat Transfer, Single Port Igniter

$P_{igniter} = 122.4 \text{ atm (1800 psi)}$

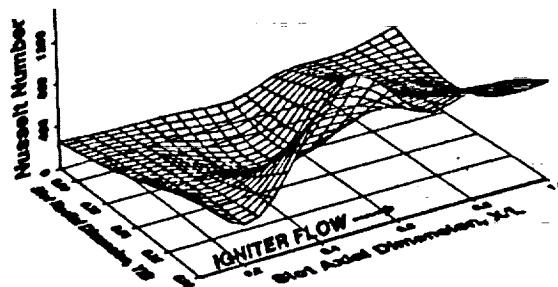


Fig. II-51 Igniter 1 (1800 psi)

Measured Heat Transfer, 22.5° Igniter

$P_{igniter} = 1800 \text{ psi}$

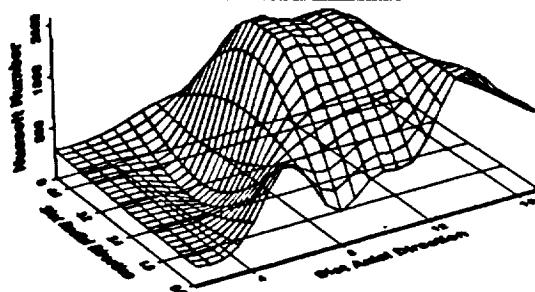


Fig. II-52 Igniter 2 (1800 psi)

Measured Heat Transfer, 45° Igniter

$P_{igniter} = 122.4 \text{ atm (1800 psi)}$

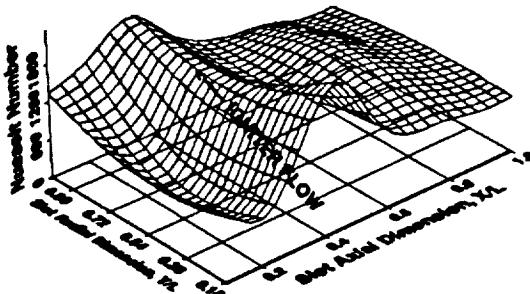


Fig. II-53 Igniter 3 (1800 psi)

References

1. Conover, G. H., Jr., "Cold-Flow Studies of Igniter Plume Flow Fields and Heat Transfer," Final Report, NASA Grant No. NGT-01-003-800, Auburn University, June 1984.
2. Schetz, J. A., Hewitt, P. A., and Thomas, R., "Swirl Combustion Flow-Visualization Studies in a Water-Tunnel," Journal of Spacecraft and Rockets, Vol. 20, No. 6, Nov-Dec., 1983, pp.574-582..
3. Schetz, J. A., Guruswamy, J., and Marchman, J. F., III, "Effects of an S-Inlet on the Flow in a Dump Combustor," Journal of Spacecraft and Rockets, March-April, 1985, pp. 221-224.
4. Schetz, J. A., Sebba, F., and Thomas, R. H., "Flow-Visualization Studies of a Solid Fuel Ramjet Combustor Using a New Material-Polyaphrone," 22nd Joint Army, Navy, NASA, Air Force Combustion Meeting, October, 1985.
5. Caveny, L. H., and Kuo, K. K., "Ignition Transients of Large Segmented Rocket Boosters," April 1976, NASA Contractor Report CR-1501162, NASA George C. Marshall Space Flight Center.
6. Simon, E., "The George C. Marshall Space Flight Center's 14x14-Inch Trisonic Wind Tunnel Technical Handbook," NASA TMX-53185, December, 1964.
7. Ciucci, A., Jenkins, R. M. and Foster, W. A., Jr., "Numerical Analysis of Ignition Transients in Solid Rocket Motors," AIAA Paper 91-2426, 27th AIAA/SAE/ASME Joint Propulsion Conference, Sacramento, California, June, 1991.
8. Ciucci, A., "Numerical Investigation of the Flow Field in the Head-End Star Grain Section of a Solid Rocket Motor During Ignition Transients," Ph.D. Dissertation, Auburn University, December, 1991.
9. Ciucci, A., Jenkins, R. M. and Foster, W. A., Jr., "Analysis of Ignition and Flame Spreading in the Space Shuttle Head-End Star Grain," AIAA Paper 92-3272, 28th AIAA/SAE/ASME/ASEE Joint Propulsion Conference and Exhibit, Nashville, Tennessee, July 6-8, 1992.

III. THEORETICAL/NUMERICAL ANALYSIS

Introduction

The ignition transient of an SRM employing a pyrogen igniter can be defined as the time interval from the onset of the igniter flow to the time a quasi-steady flow develops. The starting transient is traditionally divided into three phases: the induction interval, or ignition lag; flame spreading; and chamber filling. The induction interval begins when the igniter flow is initiated and ends when a point on the propellant surface reaches a critical ignition temperature, and a flame first appears. The flame spreading phase follows, ending when the entire propellant surface is ignited. Following this is the chamber filling phase, during which rapid chamber pressurization occurs due to the energy and mass addition from the burning propellant. A peak pressure may occur, followed by a pressure decrease towards an equilibrium value, attained when mass production by the propellant equals the mass outflow from the motor nozzle. Numerous studies have been directed at the analysis of SRM ignition transient phenomena. As discussed by Peretz, *et al.*¹, these analyses can generally be categorized into three groups: (1) lumped chamber parameter, $P(t)$ models²⁻⁷; (2) one dimensional, quasi-steady flow, $P(x)$ models^{8,9}; and (3) temporal and one-dimensional flow field, $P(x,t)$ models.^{1,10,11} The simplest analyses fall into the first group; a uniform chamber pressure is assumed and an equation for dP_{chamber}/dt is derived and integrated to obtain a pressure-time trace. The flame spreading speed is assumed to be a known constant. In $P(x)$ type models, flow property distributions are considered at each instant of time and one-dimensional steady state conservation equations are solved along the motor axis. As in the $P(t)$ type models, the flame spreading speed is not part of the solution but rather must be input. In $P(x,t)$ type models both spatial and temporal property variations are considered. A series of control volume increments are assumed along the motor axis and a set of time dependent one-dimensional conservation equations are solved. The flame spreading speed can be obtained as part of the solution if convective heat transfer to the propellant grain is taken into account. A widely used model of this type is that developed by Caveny and Kuo.¹⁰

Jasper Lal, *et al.*¹², have recently developed a one-dimensional model which takes canted pyrogen igniters into account by modifying the heat transfer analysis to include direct igniter plume impingement on the solid propellant surface. Even more recently, Johnston¹³ has presented a numerical procedure for the analysis of internal flows in a solid rocket motor wherein an unsteady, axisymmetric solution of the Euler equations is combined with simple convective and radiative fluid heat transfer models and an unsteady one-dimensional heat conduction solution for the propellant grain. Flow in the star grain slots is not directly calculated. Instead, burn rate constants are adjusted to account for the variable area in the star grain region. Results are presented for a Titan 5½ segment SRM, a Titan 7 segment SRM, and the Space Shuttle SRM. Of these three motors, the Space Shuttle

SRM has the more pronounced axial grooves in the star grain, and agreement of Johnston's model with pressure-time data in the head-end star grain section of this motor is acknowledged to be poor.

In general, it can be argued that predictions agree quantitatively well with test data for motors such as those used on the Space Shuttle, with the exception of the time period which directly involves burning of the head-end star grain segment. It may be argued that discrepancies arise primarily from three factors: (1) the flow field is usually assumed to be one dimensional; (2) the star geometry in the head-end segment is approximated by variations in port area and burning perimeter of the grain; and (3) the igniter flow field is not taken into account. The present analysis seeks to address these issues.

Conservation Equations

In this investigation, the Space Shuttle solid rocket motor (SRM) is taken as the reference motor design. It is characterized by a large length-to-diameter ratio and by a small port-to-nozzle throat area ratio. The reference motor is divided into four segments, as shown in Fig. III-1(a). The head-end, star shaped region of the solid propellant grain contains eleven slots; a cross section of the head-end segment is shown in Fig. III-1(b).

The flow field to be analyzed is extremely complex; it is unsteady, multi-dimensional, turbulent, and compressible. Further, the flow field is divided into a supersonic core region, defined by the expanding gases from the igniter, and a subsonic region inside the star slot. The two-dimensional, unsteady, compressible Navier-Stokes equations, neglecting body forces and heat source terms, are employed. The flow field is described by adopting a cylindrical coordinate system for the port region from the motor centerline to the star grain tips, and a rectangular cartesian coordinate system for the region of the flow inside the star slot.

Although the flow-field is three dimensional in the head-end section of the motor, the star grain cross sectional shape of the propellant segment implies that a certain number of planes of symmetry exist, so that it is possible to restrict the domain to a single sector, as shown in Fig. III-2. The two-dimensional Navier-Stokes equations are obtained by averaging the full, three-dimensional Navier-Stokes equations (with a formal integration) in a direction perpendicular to the plane of symmetry of the star slot. Averaging is carried out along the azimuthal coordinate, θ , in the port region, and in the z-direction inside the star slot. The calculated flow field variables thus represent an average value in the domain considered (area shaded in Fig. III-2).

The governing equations can most conveniently be solved in dimensionless form. For the flow under investigation, no "natural" free stream parameters exist. Consequently, the values of the igniter exit parameters at the condition of maximum igniter mass flow have been chosen as reference conditions, with the distance from the motor centerline to the bottom of the slot taken as the

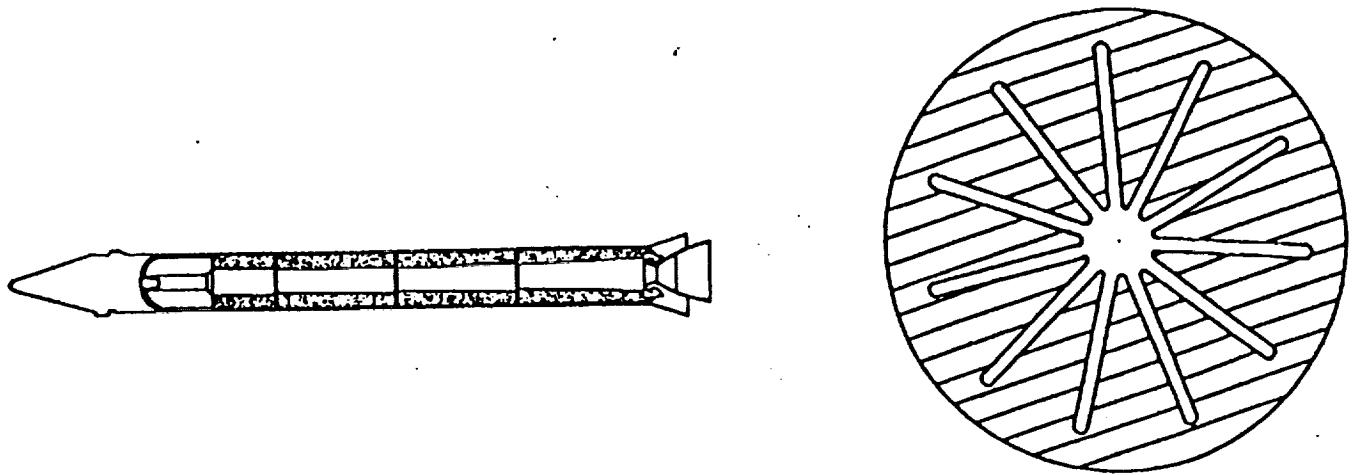


Figure III-1: (a) Space Shuttle SRM; (b) SRM Star Grain Cross Section

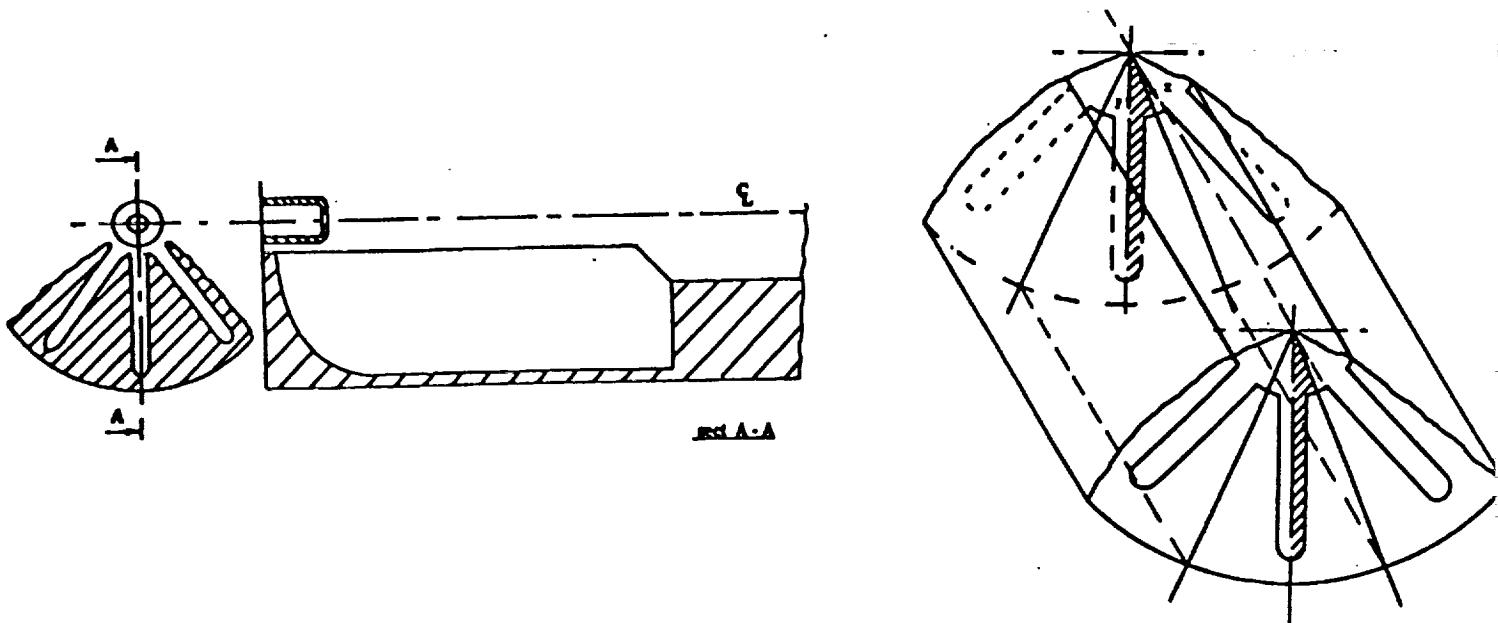


Figure III-2: SRM Star Grain Calculation Domain

reference length. It should be noted that the igniter mass flow is itself a function of time; this will be discussed in more detail later.

The dimensionless governing equations then take the following form:

continuity:

$$\frac{\partial \rho^*}{\partial t^*} + \frac{\partial \rho^* u^*}{\partial x^*} + \frac{\partial \rho^* v^*}{\partial y^*} = S_c \quad \text{III-1}$$

x-momentum:

$$\frac{\partial \rho^* u^*}{\partial t^*} + \frac{\partial \rho^* u^{*2}}{\partial x^*} + \frac{\partial \rho^* u^* v^*}{\partial y^*} = - \frac{\partial p^*}{\partial x^*} + \frac{1}{Re} \frac{\partial \mu^*}{\partial x^*} \left[\frac{4}{3} \frac{\partial u^*}{\partial x^*} - \frac{2}{3} \frac{\partial v^*}{\partial y^*} \right] \quad \text{III-2}$$

$$+ \frac{1}{Re} \frac{\partial \mu^*}{\partial y^*} \left[\frac{\partial u^*}{\partial y^*} + \frac{\partial v^*}{\partial x^*} \right] + \frac{1}{Re} \mu^* \left[\frac{4}{3} \frac{\partial^2 u^*}{\partial x^{*2}} + \frac{\partial^2 u^*}{\partial y^{*2}} + \frac{1}{3} \frac{\partial^2 v^*}{\partial x^* \partial y^*} \right] + S_{xu}$$

y-momentum:

$$\frac{\partial \rho^* v^*}{\partial t^*} + \frac{\partial \rho^* u^* v^*}{\partial x^*} + \frac{\partial \rho^* v^{*2}}{\partial y^*} = - \frac{\partial p^*}{\partial y^*} + \frac{1}{Re} \frac{\partial \mu^*}{\partial y^*} \left[\frac{4}{3} \frac{\partial v^*}{\partial y^*} - \frac{2}{3} \frac{\partial u^*}{\partial x^*} \right] \quad \text{III-3}$$

$$+ \frac{1}{Re} \frac{\partial \mu^*}{\partial x^*} \left[\frac{\partial u^*}{\partial y^*} + \frac{\partial v^*}{\partial x^*} \right] + \frac{1}{Re} \mu^* \left[\frac{4}{3} \frac{\partial^2 v^*}{\partial y^{*2}} + \frac{\partial^2 v^*}{\partial x^{*2}} + \frac{1}{3} \frac{\partial^2 u^*}{\partial x^* \partial y^*} \right] + S_{yu}$$

energy:

$$\frac{\partial \rho^* e}{\partial t^*} + \frac{\partial \rho^* u^* e^*}{\partial x^*} + \frac{\partial \rho^* v^* e^*}{\partial y^*} = \frac{1}{\gamma - 1} \frac{1}{Pr Re M^2} \left[\frac{\partial}{\partial x^*} \left(K^* \frac{\partial T^*}{\partial x^*} \right) + \frac{\partial}{\partial y^*} \left(K^* \frac{\partial T^*}{\partial y^*} \right) \right] - p^* \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} \right) + \frac{2}{Re} \mu^* \left[\left(\frac{\partial u^*}{\partial x^*} \right)^2 + \left(\frac{\partial v^*}{\partial y^*} \right)^2 \right] + \quad \text{III-4}$$

$$\frac{1}{2} \left(\frac{\partial u^*}{\partial y^*} + \frac{\partial v^*}{\partial x^*} \right)^2 + S_e$$

equations of state:

$$p^* = \frac{\rho^* T^*}{\gamma M^2} \quad \text{III-5}$$

$$e^* = \frac{T^*}{\gamma(\gamma-1)M^2} \quad \text{III-6}$$

The conservation equation source terms are given by:

$$s_c = -\frac{\rho^* v^*}{y^*} \quad \text{III-7}$$

$$s_{xx} = -\frac{\rho^* u^* v^*}{y^*} - \frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial x^*} \frac{v^*}{y^*} + \frac{1}{Re} \frac{\mu^*}{y^*} \left(\frac{1}{3} \frac{\partial v^*}{\partial x^*} + \frac{\partial u^*}{\partial y^*} \right) \quad \text{III-8}$$

$$s_{yy} = -\frac{\rho^* v^* u^*}{y^*} - \frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial y^*} \frac{v^*}{y^*} + \frac{4}{3} \frac{1}{Re} \frac{\mu^*}{y^*} \left(\frac{\partial v^*}{\partial y^*} - \frac{v^*}{y^*} \right) \quad \text{III-9}$$

$$s_e = -\frac{\rho^* e^* v^*}{y^*} + \frac{1}{\gamma-1} \frac{1}{Pr} \frac{1}{Re} \frac{K^*}{M^2} \frac{\partial T^*}{\partial y^*} - p^* \frac{v^*}{y^*} + \frac{2}{Re} \mu^* \left(\frac{v^*}{y^*} \right)^2 - \frac{2}{3} \frac{1}{Re} \mu^* \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} + \frac{v^*}{y^*} \right)^2 \quad \text{III-10}$$

for the circular port region of the computational domain. For the slot itself, these terms are given by:

$$s_c = 2\rho_p \frac{r^*}{b} \quad \text{III-11}$$

$$S_{xz} = -\frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial x^*} \frac{\partial w^*}{\partial z^*} + \frac{2}{b^*} \frac{1}{Re} (\mu^* \frac{\partial u^*}{\partial z^*}), \quad III-12$$

$$S_{yz} = -\frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial y^*} \frac{\partial w^*}{\partial z^*} + \frac{2}{b^*} \frac{1}{Re} (\mu^* \frac{\partial v^*}{\partial z^*}), \quad III-13$$

$$S_e = \frac{2}{Re} \mu^* \left[\left(\frac{\partial w^*}{\partial z^*} \right)^2 + \frac{1}{2} \left(\frac{\partial u^*}{\partial z^*} \right)^2 + \left(\frac{\partial v^*}{\partial z^*} \right)^2 \right] \\ III-14$$

$$- \frac{2}{3} \frac{1}{Re} \mu^* \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} + \frac{\partial w^*}{\partial z^*} \right)^2 + A$$

where

$$A = -2 \rho_p \cdot \frac{r^*}{b^*} e_{gp}^* \quad (grain burning) \quad III-15$$

$$A = \frac{1}{\gamma-1} \frac{1}{Pr} \frac{1}{Re} \frac{2}{M^2} \frac{b^*}{b} \left(K^* \frac{\partial T^*}{\partial z^*} \right) \quad (no grain burning) \quad III-16$$

Turbulence Model

The low Reynolds number form of the two-equation, $k-\epsilon$ model developed by Jones and Launder^{14,15} is used in the present analysis. This model employs two variables, the kinetic energy of turbulent velocity fluctuations, k , and the rate of dissipation of kinetic energy, ϵ , from which the turbulent viscosity, μ_t , is calculated. The turbulent thermal conductivity, K_t , is obtained by assuming a constant value of the turbulent Prandtl number, $\sigma_t = 0.91$.¹⁶ The turbulent kinetic energy and energy dissipation rate are determined from the solution of the following differential equations:

k -equation:

$$\frac{\partial \rho^* k}{\partial t^*} + \frac{\partial \rho^* u^* k}{\partial x^*} + \frac{\partial \rho^* v^* k}{\partial y^*} = \frac{1}{Re} \left\{ \frac{\partial}{\partial x^*} \left[\left(\mu_t^* + \frac{\mu_t^*}{\sigma_k} \right) \frac{\partial k}{\partial x^*} \right] + \frac{\partial}{\partial y^*} \left[\left(\mu_t^* + \frac{\mu_t^*}{\sigma_k} \right) \frac{\partial k}{\partial y^*} \right] \right\}$$

$$+ \frac{1}{Re} P_1 + P_2 - \frac{1}{Re} \rho^* \epsilon - \frac{2}{Re} \mu^* \left[\left(\frac{\partial k}{\partial x^*} \right)^{\frac{1}{2}} + \left(\frac{\partial k}{\partial y^*} \right)^{\frac{1}{2}} \right]^2 + S_k \quad III-17$$

ϵ -equation:

$$\begin{aligned} \frac{\partial \rho^* \epsilon}{\partial t^*} + \frac{\partial \rho^* u^* \epsilon}{\partial x^*} + \frac{\partial \rho^* v^* \epsilon}{\partial y^*} &= \frac{1}{Re} \left\{ \frac{\partial}{\partial x^*} \left[\left(\mu_t^* + \frac{\mu_t^*}{\sigma_\epsilon} \right) \frac{\partial \epsilon}{\partial x^*} \right] + \frac{\partial}{\partial y^*} \left[\left(\mu_t^* + \frac{\mu_t^*}{\sigma_\epsilon} \right) \frac{\partial \epsilon}{\partial y^*} \right] \right\} \\ &+ \frac{1}{Re} C_1 \frac{\epsilon}{k} P_1 + C_1 \frac{\epsilon}{k} P_2 - \frac{1}{Re} C_2 \rho^* \frac{\epsilon^2}{k} - \frac{2}{Re} \frac{\mu_t^* \mu_t^*}{\rho^*} \left[\left(\frac{\partial^2 u^*}{\partial x^{*2}} \right)^2 + \left(\frac{\partial^2 u^*}{\partial y^{*2}} \right)^2 \right] \\ &- \frac{2}{Re} \frac{\mu_t^* \mu_t^*}{\rho^*} \left[\left(\frac{\partial^2 v^*}{\partial x^{*2}} \right)^2 + \left(\frac{\partial^2 v^*}{\partial y^{*2}} \right)^2 + 2 \left(\frac{\partial^2 u^*}{\partial x^* \partial y^*} \right)^2 + 2 \left(\frac{\partial^2 v^*}{\partial x^* \partial y^*} \right)^2 \right] + S_\epsilon \quad III-18 \end{aligned}$$

where

$$P_1 = \mu_t^* \left\{ \frac{4}{3} \left[\left(\frac{\partial u^*}{\partial x^*} \right)^2 + \left(\frac{\partial v^*}{\partial y^*} \right)^2 - \frac{\partial u^*}{\partial x^*} \frac{\partial v^*}{\partial y^*} \right] + \left(\frac{\partial u^*}{\partial y^*} + \frac{\partial v^*}{\partial x^*} \right)^2 \right\} \quad III-19$$

and

$$P_2 = -\frac{2}{3} \rho^* k \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} \right) \quad III-20$$

The source terms are given by

$$S_k = -\frac{\rho^* v^* k}{y^*} + \frac{1}{Re y^*} \left(\mu_t^* + \frac{\mu_t^*}{\sigma_k} \right) \frac{\partial k}{\partial y^*} - \frac{1}{Re} \frac{2}{3} \mu_t^* \frac{v^*}{y^*} \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} \right) \quad III-21$$

$$S_\epsilon = -\frac{\rho^* v^* k}{y^*} + \frac{1}{Re y^*} \left(\mu_t^* + \frac{\mu_t^*}{\sigma_\epsilon} \right) \frac{\partial \epsilon}{\partial y^*} - \frac{1}{Re} \frac{2}{3} C_1 \frac{\epsilon}{k} \mu_t^* \frac{v^*}{y^*} \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} \right) \quad III-22$$

in the circular port region of the domain, and by

$$S_k = P_v \quad III-23$$

$$S_\epsilon = C_1 \frac{\epsilon}{k} P_v \quad III-24$$

$$P_v = \frac{1}{Re} \mu_t \cdot \left\{ \frac{4}{3} \left[\left(\frac{\partial w^*}{\partial z^*} \right)^2 - \frac{\partial w^*}{\partial z^*} \frac{\partial u^*}{\partial x^*} - \frac{\partial w^*}{\partial z^*} \frac{\partial v^*}{\partial y^*} \right] + \left(\frac{\partial u^*}{\partial z^*} \right)^2 + \left(\frac{\partial v^*}{\partial z^*} \right)^2 \right\}$$

III-25

$$- \frac{2}{3} \rho^* k \frac{\partial w^*}{\partial z^*}$$

for the portion of the domain within the slot.

Heat Transfer Relations

The goal of the present analysis is to examine the interaction between the igniter plume, the developing flow field within the head-end star grain slots, and the rate of flame spread over the grain surface. This, in turn, may provide insight into the appropriateness of a particular grain design or a particular igniter design for a given SRM. Many previous analyses utilize a convection heat transfer model, while others, such as Johnston¹³, utilize a combined convection-radiation model. The present analysis utilizes a simple convection-only model developed by Kays and Leung¹⁷, and used previously to correlate heat transfer within the O-ring gap of the Space Shuttle nozzle-to-case joint¹⁸, given by

$$Nu = \frac{0.152 Re^{0.9} Pr}{0.833 [2.25 \ln(0.114 Re^{0.9}) + 13.2 Pr - 5.8]} \quad III-26$$

where the Reynolds number is based on the hydraulic diameter of a single star grain slot. Fluid properties are based on a "reference temperature" denoted as¹⁹

$$T_R = T + 0.5 (T_{wall} - T) + 0.22 (T_{wa} - T) \quad III-27$$

and the adiabatic wall temperature is calculated from

$$T_{wa} = T + \sqrt[3]{Pr} \frac{V^2}{2C_p} \quad III-28$$

The propellant burning rate is assumed to be of the form

$$r = r_{ref} \left(\frac{P}{P_{ref}} \right)^n \exp [a_p (T_{p_i} - T_{ref})] \quad III-29$$

The constants appearing in Eq.(27) are defined in Table III-1. Erosive burning is assumed to be negligible during the very early portion of the ignition transient.

III-8

Table III-1. Constants used in burning rate law

Constant	Value	Units
r_{ref}	0.01078	$m \ sec^{-1}$
P_{ref}	6898.2	KPa
n	0.35	-
σ_p	0.002	K^{-1}
T_{ref}	300	K

In order to determine when a given element of the solid propellant reaches a critical ignition temperature, one must know the surface temperature of the solid grain. This, in turn, depends on the amount of heat transferred to the grain from the hot gases.

The grain is considered to be a semi-infinite slab whose temperature is initially uniform. Heat transfer to the slab is assumed to be one-dimensional. Thus

$$\frac{\partial T_p}{\partial t} = \alpha_p \frac{\partial^2 T_p}{\partial z^2} \quad III-30$$

with the boundary conditions

$$T_p(t, \infty) = T_{p_i} ; \quad \frac{\partial T_p(t, 0)}{\partial z} = -\frac{h_c}{K_p} [T - T_p(t, 0)] \quad III-31$$

and the initial condition

$$T_p(0, z) = T_{p_i} \quad III-32$$

The assumption of one dimensional conduction heat transfer implies that ignition of adjacent grain surface elements is attributed to direct heat transfer from the hot gas only. This appears to be a reasonable assumption because of the low thermal conductivity of the solid propellant. Assumed physical properties of the propellant grain material are given in Table III-2.

Table III-2. Solid propellant properties

Property	Value	Units
ρ_p	1758	Kg m^{-3}
K_p	0.4605	$\text{W m}^{-1} \text{K}^{-1}$
$(c_p)_p$	1256	$\text{J Kg}^{-1} \text{K}^{-1}$
$T_p, \text{critical}$	850	K

The initial propellant temperature is assumed to be $T_{pi} = 298^\circ\text{K}$. The (slot) wall shear stress must be approximated in order to provide closure for the governing equations. A velocity profile across the slot width (in the z -direction) is assumed for the velocity components u and v . Pai's²⁰ polynomial form of the velocity profile for a flat duct is employed. For example, the u component of velocity is

$$\frac{u}{u_{\max}} = 1 - \frac{p-q}{p-1} \left(\frac{z}{b/2} \right)^2 - \frac{q-1}{p-1} \left(\frac{z}{b/2} \right)^{2p} \quad \text{III-33}$$

where u_{\max} is the velocity at the symmetry plane of the duct (slot), $p = 11.06$, and $q = 16^{20}$. Although Eq. (III-33) applies specifically to fully developed flow, it is used here to account for wall shear stress effects even though the flow is time variant. It should be noted that frictional effects at the slot walls are accounted for only in the unignited portion of the propellant; they are neglected for the ignited portion because of the strong transverse velocity component due to mass injection from the burning propellant.

Finally, the transverse velocity component $w(z)$ is assumed to be linear, with $w = 0$ at the slot plane of symmetry; w at the slot wall can be determined from the known burning rate. Hence,

$$w_{\text{wall}} = \frac{\rho_p x}{p} ; \quad \frac{\partial w}{\partial z} = -\frac{2}{b} w_{\text{wall}} \quad \text{III-34}$$

Numerical Technique

The numerical solution of the equations of motion is obtained utilizing the explicit, time-dependent, predictor-corrector finite difference method developed by MacCormack.²¹ This method has been widely used for the numerical solution of a variety of fluid

dynamics problems, including those containing mixed subsonic-supersonic flow regions, as in the present investigation. MacCormack's predictor corrector technique is well described in the existing literature.^{22,23} In the present investigation, an explicit, fourth order numerical dissipation scheme has been introduced into the set of equations to damp numerical oscillations induced by the severe gradients in the flow field associated with the developing igniter flow. The fourth-order damping scheme introduced by Holst²⁴ and modified by Berman²⁵ and Kuruvila²⁶ is employed. This scheme involves certain free adjustable parameters, C_x and C_y , usually referred to as damping or dissipation coefficients. It is recommended in the literature^{22,23} that C_x , C_y be such that $0 \leq C_x, C_y \leq 0.5$.

A uniform rectangular grid is employed in the portion of the domain that represents the head-end section, which includes the star slot segment plus a distance equal to the port diameter. For this region a 92x63 grid has been used, as shown in Figure III-3. A grid of unequal spacing in the x-direction is used in the extended portion of the domain from the head-end segment to the motor exit. The scheme of Cebeci and Smith,²⁷ in which grid spacing is increased by a fixed percentage from an initial point, is utilized.

Initial and Boundary Conditions

In time marching problems, the initial conditions should in no way affect the steady state results. In theory, any initial conditions can be chosen. Of course, for an arbitrary set of initial conditions, the transient has no physical meaning and only the steady state condition is a meaningful representation of the flow field.

Since in this study the starting transient is of primary importance, the initial conditions must reflect the actual values of the flow field variables at time $t = 0$. Therefore, both components of the velocity have been set equal to zero, $u = v = 0$, and ambient values have been chosen for pressure and temperature everywhere in the flow field. Density and internal energy are obtained from the equation of state. The turbulent kinetic energy, k , and its dissipation rate, ϵ , are also zero initially. However, imposing this value would lead to a singularity in the k and ϵ equations; thus, a very small value has been assigned to these two variables.

Boundary conditions must also be specified for all of the dependent variables, u , v , p , e , k , ϵ , along with corresponding values of p and T . Of primary importance is the specification of the time variant conditions from the developing igniter flow at the inflow boundary of the calculation domain. The conditions specified are those from the single-port igniter used in the Space Shuttle SRM. The igniter mass flow vs. time trace is shown in Figure III-4, using data from Ref. 28. From the known igniter mass flow rate vs. time trace and the geometry of the igniter nozzle, the values of the flow variables u , p , p , T at the igniter nozzle

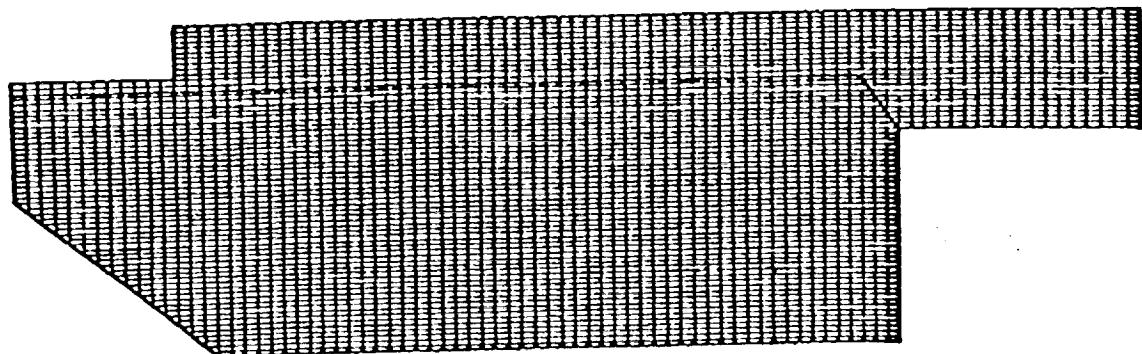


Figure III-3: Computational Grid

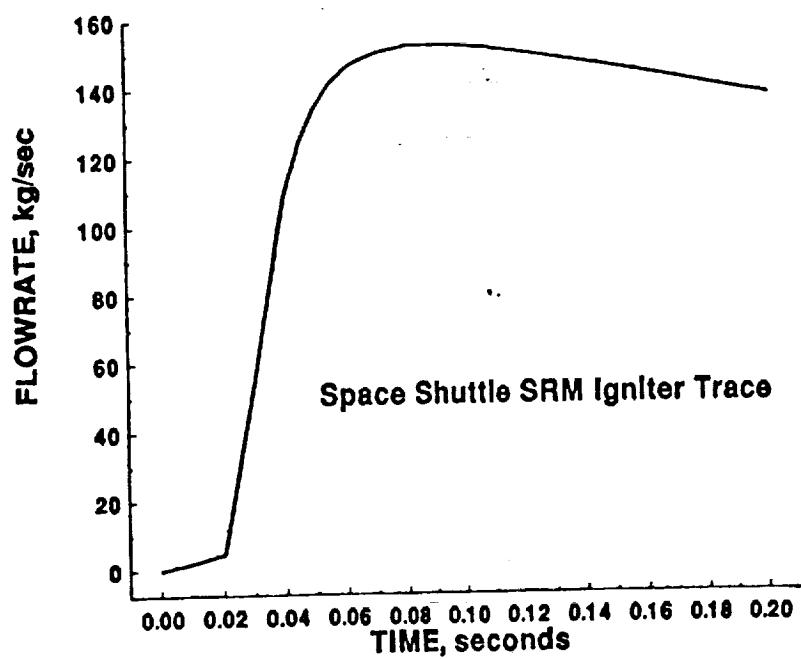


Figure III-4: Space Shuttle SRM Igniter Flow vs. Time Trace

exit are calculated using one-dimensional nozzle flow theory; the radial velocity is assumed to be zero, $v=0$, and the internal energy is calculated from the known temperature. Note that the numerical method employed is a shock capturing technique, so that the shock waves associated with the igniter plume are embedded in the solution. Inflow boundary conditions for k and ϵ must be also specified. Since no experimentally measured profiles are available, the turbulent kinetic energy is assumed to be equal to a small percentage of the freestream kinetic energy, and the energy dissipation rate is taken from an empirical relation used, for instance, in Ref. 29. Along most solid walls of the computational domain the "no slip" condition is enforced, that is, $u = v = 0$; also, due to the low Reynolds number form of the two-equation turbulence model, $k = 0$ and $\epsilon = 0$. For these walls the temperature boundary condition is specified by assuming an adiabatic wall. The exception is the cylindrical port portion of the domain downstream of the star slot section. For this region, the wall temperature is determined in the same manner as in the upstream star slot, and a "blowing wall" ($v \neq 0$) condition is imposed to account for mass addition due to burning. The top boundary of the computational domain represents the centerline of the motor, so that a symmetry boundary condition is enforced here.

The outflow boundary at the aft end of the motor represents the final location where values of the dependent variables must be specified. The approach adopted is to place a diaphragm at the motor exit, immediately preceding a fictitious "nozzle". When the pressure differential across the diaphragm reaches some nominal value, say 1 atm, the diaphragm bursts. The nozzle is assumed to fill instantaneously. Continuous checks are made on the amount of flow approaching the exit and the amount of flow the "nozzle" could pass given the existing upstream conditions. A simple one dimensional time-dependent mass conservation calculation is then utilized to provide boundary conditions for the next time step. It should be noted that, for the first 200 msec or so following igniter firing, the downstream boundary condition is not as important as it would be later in the ignition transient.

Sample Results

Cold Flow

After verification of the overall flow model and computational technique by comparison with known solutions or measurements for problems such as supersonic laminar flow over a flat plate³⁰ and developing turbulent flow in a pipe³¹, attention was turned to the igniter plume as a measure of the model's ability to predict more complex flow fields. Experimental data obtained both by Conover³² and in the series of tests described herein were utilized. Schlieren photographs of the exhaust plume(s) were taken for each igniter at various values of upstream total pressure. In particular, the Space Shuttle SRM igniter is a single port, conical nozzle configuration with an area ratio of 1.428 and a half angle of 27.2 degrees, with a design exit Mach number of 1.79. Figure

III-5 compares the calculated plume Mach number contours for an igniter total pressure of 102 atm (1500 psi) and a total temperature of 316 °K with the Schlieren photograph of the plume at the same conditions. A qualitative comparison indicates that the results are quite satisfactory, although the shock wave appears to be smeared over several grid points.

Next, calculations were performed for the cold flow (no burning) flow field within the star slot of the head-end section. The computed results can be compared with the oil smear data obtained in the present cold flow tests. Both the case of a single port igniter and a multi-port (canted) igniter are considered. The multi-port igniter exhaust jet is aligned with the slot, at an angle of 45 degrees with respect to the motor centerline. Typical results are shown in Figures III-6:III-9. Figures III-6 and III-7 compare results for an igniter pressure of 6.8 atm (100 psi), while Figures III-8 and III-9 compare results for 34 atm (500 psi). The igniter (not shown) is located near the upper right corner of the slot, exhausting from right to left.

Hot Flow

Attention was next turned to the problem of heat transfer and the calculated progression of the burning surface of the propellant grain within the star slot. As time progresses, the propellant surface temperature rises due to heat transfer from the hot gas flowing over it. Propellant ignition is assumed to occur when the temperature of the surface reaches a critical ignition temperature (850° K). Obviously, the resulting flame spread is dependent on the heat transfer correlation assumed (eg., Eq. III-26). Figure III-10 illustrates a typical predicted burn sequence for the single port igniter presently used on the Space Shuttle SRM. First ignition of the propellant surface occurs in the vicinity of the shock located in the igniter plume, as might be expected. Interestingly, this burning progression pattern can be anticipated from calculated cold flow heat transfer contours, as can be seen in Figure III-11, which illustrates calculated Nusselt number contours for cold flow over a range of igniter pressures from 6.8 atm (100 psi) to 102 atm (1500 psi). Again, the igniter exhausts from right to left.

Figure III-12 compares the calculated head-end pressure-time trace with values obtained from measurements taken from actual motor firings¹¹, as well as with values calculated from a typical $P(x,t)$ model¹⁰. The comparison is made for the time interval $0 \leq t \leq 120$ milliseconds, over which the first propellant grain ignition occurs and flame spreading begins on the slot surface. It can be seen that the $P(x,t)$ model significantly underpredicts the head end pressure during this interval, while the present model matches the measured pressure trace with reasonable accuracy. During this period, the flame spread is fairly slow, as would be expected. A flame first appears on the propellant surface at 25 msec after the igniter firing, and at 120 msec approximately 20 percent of the star slot grain is burning. The rapid pressure rise shown by the $P(x,t)$ model at approximately 115 msec corresponds to

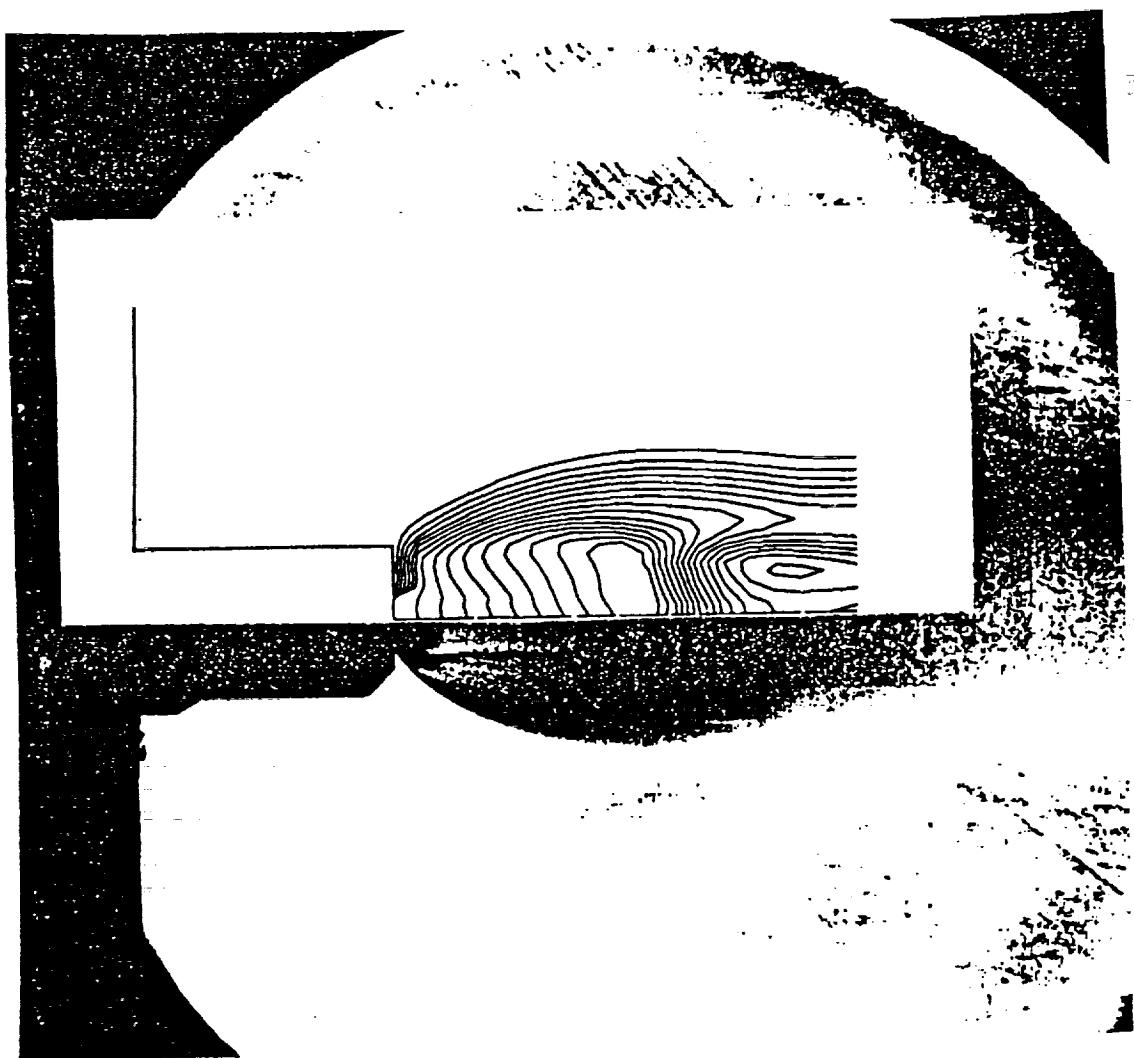
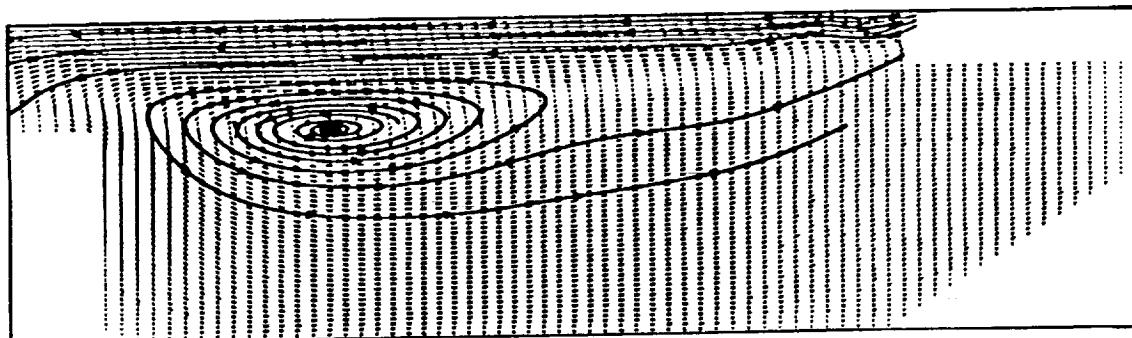


Figure III-5: Igniter Flowfield Comparison (cold-flow)



Oil Smear

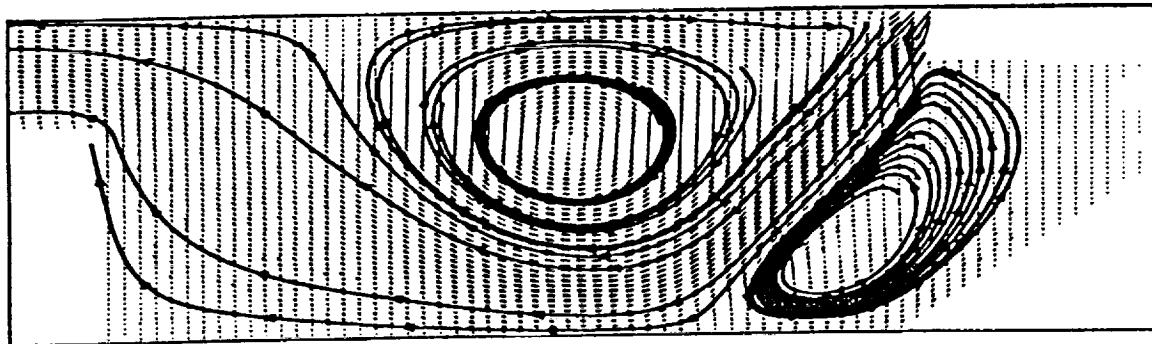


Calculated

Figure III-6: Single Port Igniter Cold-Flow, $P_{igniter} = 6.8$ atm

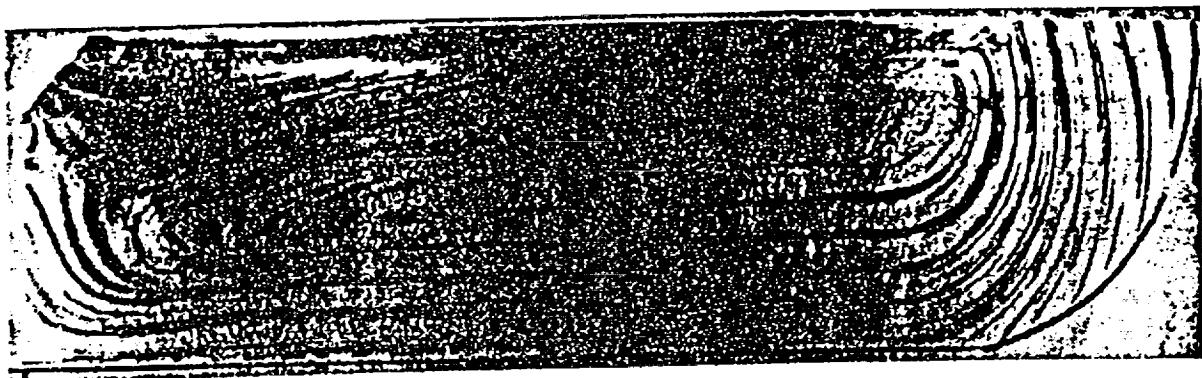


Oil Smear

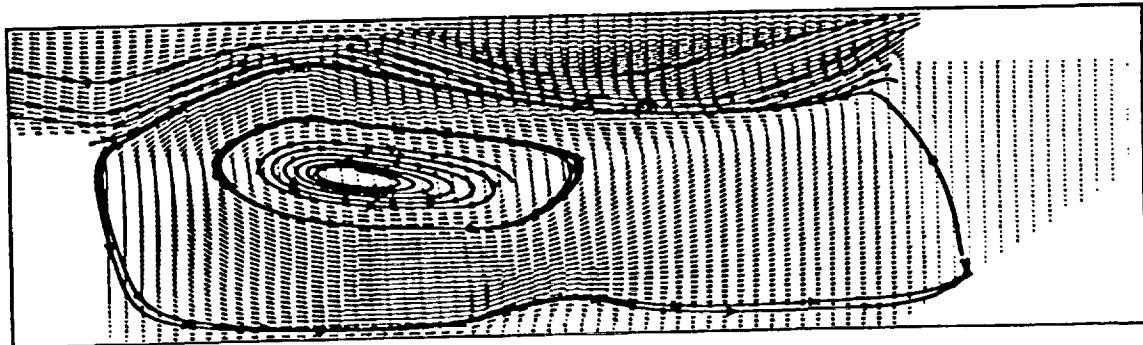


Calculated

Figure III-7: 45 Degree Igniter Cold-Flow, $P_{igniter} = 6.8$ atm



Oil Smear

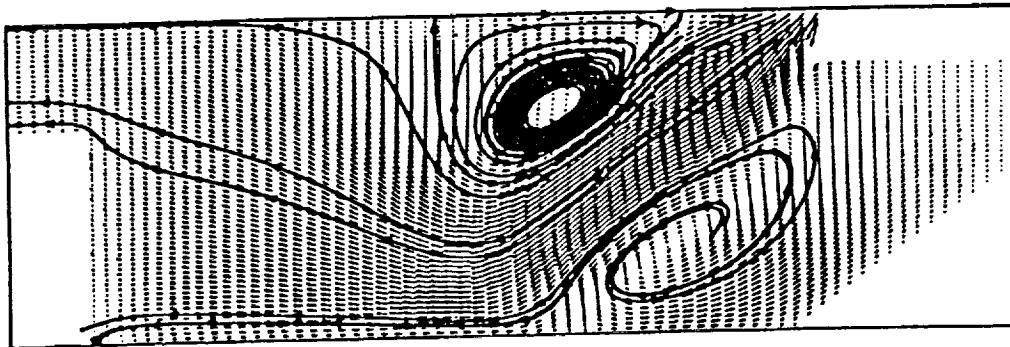


Calculated

Figure III-8: Single Port Igniter Cold-Flow, $P_{igniter} = 34$ atm



Oil Smear



Calculated

Figure III-9: 45 Degree Igniter Cold-Flow, $P_{igniter} = 34$ atm

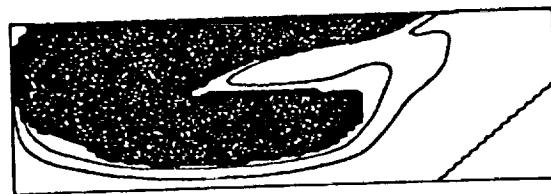
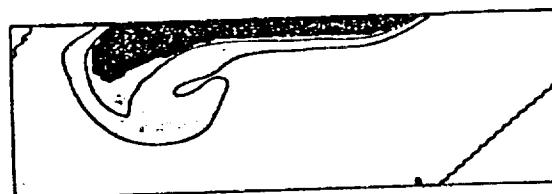
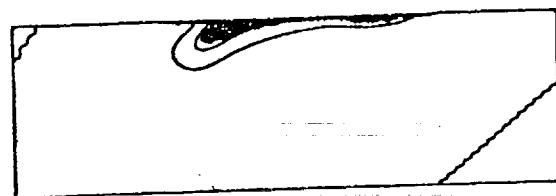
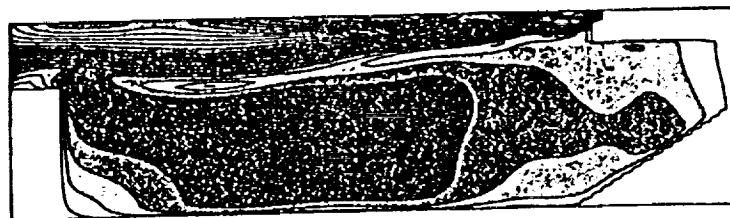
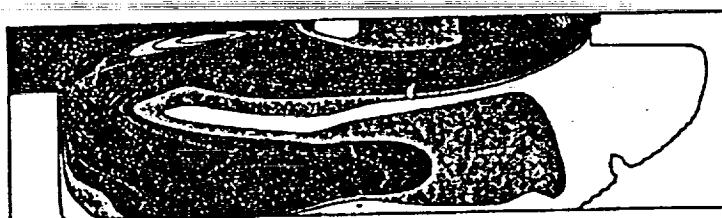


Figure III-10: Typical Slot Area Burn Sequence (predicted)

Calculated (Cold Flow) Heat Transfer Contours
Igniter Pressure = 6.8 atm (100 psi)



Calculated (Cold Flow) Heat Transfer Contours
Igniter Pressure = 34 atm (500 psi)



Calculated (Cold Flow) Heat Transfer Contours
Igniter Pressure = 102 atm (1500 psi)

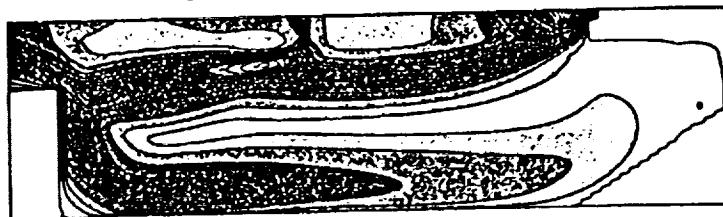


Figure III-11: Calculated Nu Contours (Cold-Flow) Single Port Igniter

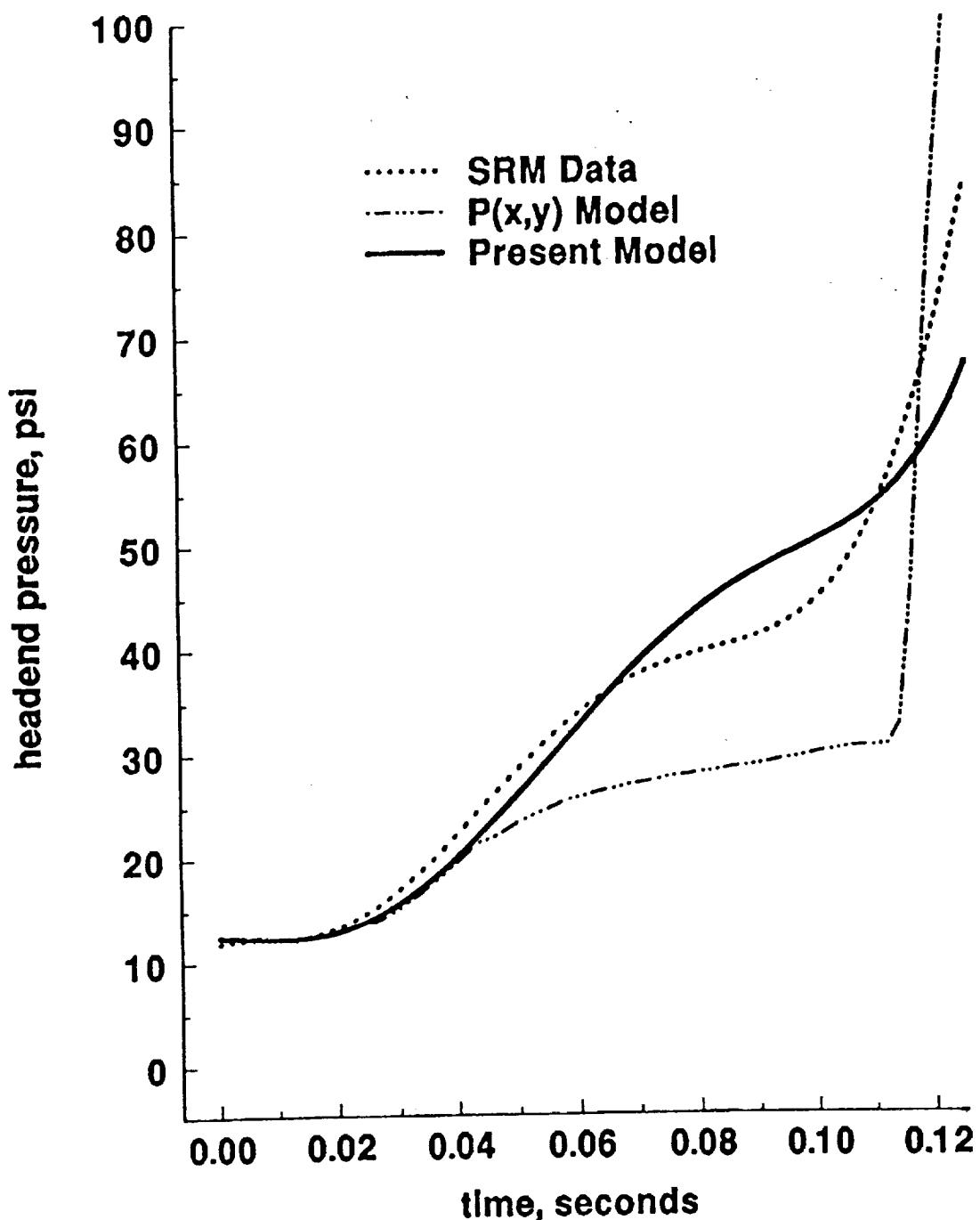


Figure III-12: (Early) Ignition Transient Head-End Pressure Rise

the ignition of the CP portion of the grain in that model. First ignition of the CP portion is predicted to be at approximately 80 msec in the present model. It should be noted that the present model tends to under-predict the pressure rise in the head-end for $t > 120$ msec. This suggests that the predicted flame spread for time $t > 120$ msec is too low. This may be due to one or more of several effects, including under-prediction of the convection heat transfer coefficient, the presence of significant radiative heat transfer, or the significance of 3-D conduction effects within the propellant grain.

References

1. Peretz, A., et. al., "Starting Transient of Solid-Propellant Rocket Motors with High Internal Gas Velocities," AIAA Journal, Vol. 11, No. 12, December 1973, pp. 1719-1727.
2. DeSoto, S., and Friedman, H.A., "Flame Spreading and Ignition Transients in Solid Grain Propellants," AIAA Journal, Vol. 3, March 1965, pp. 405-412.
3. Sforzini, R.H., and Fellows, H.L., Jr., "Prediction of Ignition Transients in Solid Propellant Rocket Motors," Journal of Spacecraft and Rockets, Vol. 7, No. 5, May 1970, pp. 626-628.
4. Sforzini, R.H., "Extension of a Simplified Computer Program for Analysis of Solid-Propellant Rocket Motors," NASA CR-129024, Auburn University, AL, April 1973.
5. Caveny, L.H., and Summerfield, M., "Micro-Rocket Impulsive Thrusters," Aerospace and Mechanical Science Rept. 1014, November 1971, Princeton, N.J.
6. Bradley, H.H., Jr., "Theory of a Homogeneous Model of Rocket Motor Ignition Transients," AIAA Preprint No. 64-127, January 1964.
7. Most, W.J., and Summerfield, M., "Starting Thrust Transients of Solid Rocket Engines," Aerospace and Mechanical Science Report No. 873, July 1969, Princeton University, N.J.
8. Threewit, T.R., Rossini, R.A., and Uecker, R.L., "The Integrated Design Computer Program and the ACP-1103 Interior Ballistics Computer Program," Rept. No. STM-180, Aerojet-General Corp., December 1964.
9. Vellacott, R.J., and Caveny, L.H., "A Computer Program for Solid Propellant Rocket Motor Design and Ballistics Analysis," ARS Preprint No. 2315-62, January 1962.

10. Caveny, L.H., and Kuo, K.K., "Ignition Transients of Large Segmented Solid Rocket Boosters," Final Report, NASA CR-150162, April 1976.
11. Caveny, L.H., "Extension to Analysis of Ignition Transients of Segmented Rocket Motors," Final Report, NASA CR-150632, January 1978.
12. Jasper Lal, C., et. al., "Prediction of Ignition Transients in Solid Rocket Motors Employing Canted Pyrogen Igniters," Journal of Propulsion and Power, Vol. 6, No. 3, May-June 1990, pp. 344-345.
13. Johnston, W.A., "A Numerical Procedure for the Analysis of the Internal Flow in a Solid Rocket Motor During the Ignition Transient Period," AIAA Paper 91-1655, AIAA 22nd Fluid Dynamics, Plasma Dynamics, & Lasers Conference, Honolulu, Hawaii, June 1991.
14. Jones, W. P., Launder, B. E., "The Prediction of Laminarization with a Two-Equation Model of Turbulence", International Journal of Heat and Mass Transfer, Vol. 15, 1972, pp. 301-313.
15. Jones, W. P., Launder, B. E., "The Calculation of Low-Reynolds-Number Phenomena with a Two-Equation Model of Turbulence", International Journal of Heat and Mass Transfer, Vol. 16, 1973, pp. 1119-1129.
16. Launder, B. E., Spalding, D. B., "The Numerical Computation of Turbulent Flows", Computer Methods in Applied Mechanics and Engineering, Vol. 3., 1974, pp. 269-289.
17. Kays, W.M., and Leung, E.Y., Int. Journal of Heat and Mass Transfer, vol.6, 1963, pp.537-557.
18. "2-D SRM Nozzle-to-Case Joint Test Data Report," Final Report, Contract No. NAS2-12037, to NASA Marshall Space Flight Center, Micro Craft Inc., August 1988.
19. Eckert, E.R.G., "Engineering Relations for Friction and Heat Transfer to Surfaces in High Velocity Flow," JAS 22, 1955, pp.585-587.
20. Kakac, et. al., Handbook of Single-Phase Convective Heat Transfer, Wiley Publishing, N.Y., 1987, p. 461.
21. MacCormack, R. W., "The Effect of Viscosity in Hypervelocity Impact Cratering," AIAA Paper 69-354, 1969.
22. Anderson, J. D., Modern Compressible Flow: with Historical Perspective, MacGraw-Hill Book Co., New York, 1982.

23. Anderson, D.A., et. al., Computational Fluid Mechanics and Heat Transfer, Hemisphere Publishing Corp., New York, 1984.
24. Holst, T. L., "Numerical Solution of Axisymmetric Boattail Fields with Plume Simulators," AIAA Paper 77-224, 1977.
25. Berman, H. A., Anderson, J. D., "Supersonic Flow over a Rearward Facing Step with Transverse Nonreacting Hydrogen Injection," AIAA Journal, Vol.21, No.12, Dec. 1983, pp.1707-1713.
26. Kuruvila, K., Anderson, J. D., "A Study of the Effects of Numerical Dissipation on the Calculation of Supersonic Separated Flows," AIAA Paper 85-0301, 1985.
27. Cebeci, T., Smith, A. M. O., Analysis of Turbulent Boundary Layers, Academic Press, New York, 1974.
28. "Modified Igniter Performance Prediction," Doc. No. TWR-16265 L211-FY88-M035, Morton Thiokol, Inc., Brigham City, Utah, 1987.
29. Chen, L., Tao C. C., "Study of the Side-Inlet Dump Combustor of Solid Ducted Rocket with Reacting Flow," AIAA Paper No. 84-1378, 1984.
30. Allen, J.S., and Cheng, S.I., "Numerical Solution of the Compressible Navier-Stokes Equations for the Laminar Near Wake," The Physics of Fluids, Vol.13, No.1, January 1970, pp.37-52.
31. Barbin, A.R., and Jones, J.B., "Turbulent Flow in the Inlet Region of a Smooth Pipe," Trans. ASME, Journal of Basic Engineering, Vol.85, 1963, pp.29-34.
32. Conover, G. H., Jr., "Cold-Flow Studies of Igniter Plume Flow Fields and Heat Transfer," Final Report, NASA Training Grant No.NGT-01-003-800, Auburn University, Jun. 1984.

IV. CAVENY PROGRAM MODIFICATIONS

To incorporate the results obtained from the experimental and analytical work modifications were made to the original solid rocket motor ignition code. These modifications proceeded along two independent paths: changing the code so that it could work with the interface program, and changing the code to improve its accuracy when used with star grain segments.

To significantly increase the accuracy of the prediction codes used by the Caveny program, two of its fundamental assumptions had to be discarded and replaced with other computational models. First, the original code assumed that the propellant grain ignited at a given temperature, that the entire segment was not burning until it reached this temperature, and that after reaching that temperature the whole segment would burn evenly. Second, rather than calculating the burning perimeters of the segments based on input motor geometries, the Caveny program accepted an equivalent effective perimeter of the segment at various stations along the grain. This effective perimeter was modeled as a circular perforated grain, and the gas dynamics equations assumed one dimensional flow in a cylinder. These assumptions led to inaccuracies for star grains.

To correct the inadequacies of the first assumption, a new model for the burning area was introduced. A system was introduced in which the fraction of the segment burning was a function of simulation time rather than local temperature. A table of percentage burning surface as a function of time could be specified for any of the segments. This is obtained from the CFD model described in Section III. A variable NBPTAB (number of burning perimeter tables) was added to the NAME namelist. The variables NBPENT, BPTIME, and BPFRAC were added to the INPUT common block, respectively representing up to thirty burning perimeter entries in a given table and up to twenty values of time and burning surface fraction per station. These tables contain discrete values of burning surface fraction as a function of time. A linear interpolation function INTERP was also added to the Caveny code to calculate intermediate values. In addition, a new model for turning on burning of the grain had to be introduced to the code as well. As previously stated, in the previous model, the grain segments were assumed to begin burning when they reached the ignition temperature.

A subroutine to calculate burning areas as functions of distance burned was also added. The original version of the Caveny program held the burning perimeter constant during the course of a run. This assumption was considered a valid approximation since the simulated burning times and burn distances would be small in comparison to the total burn time and overall size of the motor.

To correct the shortcomings of this second assumption, a new model for calculating the burning perimeter was introduced. In this model, a number of different grain geometries can be specified in the input case file. A variable NBGTAB (number of burning geometry tables) was added to the NAME namelist. The variables NBGENT and BGVAL were added to the INPUT common block, respectively representing the number of burning geometry variables in a given table and up to ten variables that have different meanings depending on the geometry model selected. A subroutine to integrate the burning rate, BDCALC, and a variable representing the total burn distance at an axial station, BDIST, were also added to the Caveny program.

The Caveny program reads a single input file that specifies all parameters for a particular case. This file consists of at least one namelist, followed by a number of tables, followed by a number of optional namelists. The program reads the first namelist, and from it modifies the default values set in the program. This namelist includes variables describing the motor geometry, simulation parameters such as time step and end time, options for printing results, etc. Following this initial namelist are the tables for igniter mass flow and motor geometry. After these tables are a number of optional NAME namelists, each of which can be used to modify any of the simulation parameters listed in NAME. These subsequent namelists are read when the Caveny program has reached the end of a run. The new set of variables is generally restricted to output parameters such as time increment, print interval, and updated end time.

The tables associated with the new burning fraction function are specified by a single integer value for NBPENT(I) (read using a 1I10 format) between 1 and 20 representing the number of burning fraction versus time pairs for that location. These pairs of values for BPTIME(I,J) and BPFRAC(I,J) (read using a 2F10.4 format) immediately follow, one per line. An example set of burning fraction tables is given in Table IV.1.

In each of the following example listings, a vertical bar and all text following it should be interpreted as comments and should not be inserted into the input files.

Table IV.1: Sample Burning Perimeter Fraction Table

```

&NAME
...
NBPTAB = 4          | 0 <= n <= 30 tables listed below    ...
&END
5
0.0000  0.0000    | 5 entries in Table #1
0.2500  0.1000    | 0% burning at time = 0.00 sec
0.5000  0.2000    |
0.7500  0.4000    |
1.0000  0.8000    | 80% burning at time = 1.00 sec
4
...

```

These tables must be placed in the input file immediately following the initial NAME namelist, but before any of the other tables (such as igniter mass flow, grain geometry, etc.) that can be specified. The number of tables provided must correspond to the number specified in NBPTAB, and the number of entries per table must correspond to the value provided for NBPENT(J). The tables associated with the new burning geometry function are specified in a single integer value for NBGENT(I) (read using a 1I10 format) between 1 and 10 representing the number of variables in the table for that location. These variables are interpreted differently by the various motor geometries that can be specified. In all cases, the first variable BGVAL(J,1) indicates the type of geometry specified at the station. The number of variables that follow and their meanings are determined by this first value. Table IV.2 shows the valid values and their meanings. Tables IV.3 through IV.7 detail the meanings of the remaining variables in the table for each type of available geometry. When possible, variable definitions are linked to corresponding quantities between various grain configurations.

Table IV.2: Caveny Code Input Tables

BGVAL(I,1)	Grain type
1.0	Circular perforated
2.0	Slotted
3.0	Star
4.0	Wagonwheel
5.0	Dogbone

Table IV.3: Circular Perforated Grain Variables

Variable	Units	Variable Meaning
BGVAL(I,1)	none	1.0 = Circular perforated grain
BGVAL(I,2)	inches	Interior diameter (D1)
BGVAL(I,3)	inches	Exterior diameter (D2)

Table IV.4: Slotted Grain Variables

Variable	Units	Variable Meaning
BGVAL(I,1)	none	2.0 = Slotted grain
BGVAL(I,2)	inches	Interior diameter (D1)
BGVAL(I,3)	inches	Exterior diameter (D2)
BGVAL(I,4)	none	Number of star points (N)
BGVAL(I,5)	inches	Slot width

Table IV.5: Star Grain Variables

Variable	Units	Variable Meaning
BGVAL(I,1)	none	3.0 = Star grain
BGVAL(I,2)	inches	Interior diameter (D1)
BGVAL(I,3)	inches	Exterior diameter (D2)
BGVAL(I,4)	none	Number of star points (N)
BGVAL(I,5)	degrees	Star half-angle

Table IV.6: Wagonwheel Grain Variables

Variable	Units	Variable Meaning
BGVAL(I,1)	none	4.0 = Wagonwheel grain
BGVAL(I,2)	inches	Interior diameter (D1)
BGVAL(I,3)	inches	Exterior diameter (D2)
BGVAL(I,4)	none	Number of star points (N)
BGVAL(I,5)	degrees	Star half-angle

Table IV.7: Dogbone Grain Variables

Variable	Units	Variable Meaning
BGVAL(I,1)	none	5.0 = Dogbone grain
BGVAL(I,2)	inches	Interior diameter (D1)
BGVAL(I,3)	inches	Exterior diameter (D2)
BGVAL(I,4)	none	Number of star points (N)
BGVAL(I,5)	degrees	Star half-angle

The routine to calculate the burning perimeters as functions of time were based on the work presented in Reference 2. A sample set of geometry description tables is given in Table IV.8.

Table IV.8: Sample Burning Geometry Table

```

&NAME
...
NBGTAB = 2
...
&END
1.0          | Table #1: CP Grain (3 vars)
54.345       |   Inner diameter (inches)
23.456       |   Outer diameter (inches)
2.0          | Table #2: Star Grain (5 vars)
54.345       |   Inner diameter (inches)
123.456      |   Outer diameter (inches)
6.0          |   Number of star points (-)
15.0         |   Star half-angle (degree)
...

```

The original Caveny program produced two types of output files: a 132-column file reflecting key simulation variables as functions of time and (where appropriate) space, and a plot file that was designed specifically to work with the Princeton University plotter.

Rather than changing the format of the default output file of the Caveny program, new routines to generate a separate plot file were added instead. Of all the simulation variables calculated and printed in the original program, only the variables listed in Table IV.9 were considered to be of interest for plotting.

Table IV.9: Variables Saved in caveny.plot File

Time and Space Dependent Variables		Time Dependent Variables	
Name	Meaning	Name	Meaning
PSIA	Pressure	PESTAP	Exit Pressure
T	Temperature	BMTOT	Mass Burned
M	Mach Number	M	Mach Number (Nozzle)
BR	Burn Rate	FLBF	Thrust
TAUB	Burn Distance	- none -	(unused)
XMSA	Mass Flow Rate	- none -	(unused)
TPS	Surface Temperature	- none -	(unused)
REP	Reynolds Number	- none -	(unused)

To make the Caveny program more maintainable and modifiable, it was restructured to take advantage of the program development features offered by the Unix platform. In its original form, the program was approximately 3500 lines of FORTRAN code in a single file. All common blocks and namelists had to be repeated in each subroutine that used them. These common blocks and namelists often had incompatible lengths and variable names substituted between different subroutines.

To make the code more readable and more easily edited, the source was broken up into separate files that contained only one subroutine each. The filenames had a basename that was the same as the subroutine they contained and a .f suffix. In addition, all common blocks and namelists were also placed in individual

files. These files were given names corresponding to the name of the common block or namelist and a .com or .nam suffix respectively. These files were then referenced in the code through INCLUDE statements. This organization insured that identical copies of the common blocks and namelists were being used by all subroutines.

This new organization offered another advantage in program development. With each subroutine placed in only one file, only the files that had been changed since the previous build had to be recompiled, rather than compiling all modules after any change. This change was further aided through use of the Unix make utility. An appropriate makefile that was suitable for the whole project was created. This program organization made modifying the program much easier. A complete listing of this makefile is provided in Appendix B.

References

1. Sforzini, R. H., "Design and Performance Analysis of Solid-Propellant Rocket Motors Using a Simplified Computer Program," NASA Contractor Report CR-129025, October, 1972.
2. Caveny, L. H., and Kuo, K. K., "Ignition Transients of Large Segmented Rocket Boosters," April 1976, NASA Contractor Report CR-1501162, NASA George C. Marshall Space Flight Center.

V. CAVENY PROGRAM INTERFACE

The interface application is designed to make running the Caveny program easier for potential users, both novice and expert. The three natural divisions of using the program were: building the input data files, executing the program, and collecting and interpreting the results it generated. The principal parts of the interface program are designed to correspond to these same three areas.

An input file editor which is included is more than a simple text processor. Ideally, it should prevent the user from creating an input file that the program could not run. In practice, this would be nearly impossible, but it could ensure that all variables fall within certain constraints. From within the interface application, the Caveny program runs in the background. The interface program automatically reads in the data generated from the most recent execution. It then be able to produces a number of plots simultaneously. These plots are dynamic, in that the user configures the variables displayed, the range of values over which they are shown, and all aspects of the plots' appearance.

The parts of the interface program that generate input files for the Caveny program are the most crucial in making the program easy to use. Configuring input files proved to be the most daunting and error-prone aspect of running the Caveny program. As such, attention was paid to providing the user with a number of features to make editing these files more convenient. The input file editor is designed to provide two critical functions: on-line documentation of all simulation variables and error- checking of the values provided for those variables. The documentation for the variables includes (but is not necessarily limited to) definitions and descriptive text, units in which the variables should be input, and default values for each. Limited error-checking is provided by comparing the input values against known valid minimum and maximum values. While this technique does not insure that a case is consistent and will run properly, it does provide a safeguard against obvious errors.

A simple text file format has been chosen to support this part of the program. A file containing all data about the variables that could be modified in the Caveny program input case files would have significant advantages over compiling this information into the interface program. Descriptive text and default values could be changed by a user without recompiling the code, and information tailored to different audiences could be maintained. More importantly, if new variables are added to the Caveny program in the future, these variables can be easily

incorporated into the input file generation part of the interface program.

The output capability of the interface program is provided to reduce the amount of data generated by the Caveny program and to present the data in a more easily recognizable and understandable fashion than simple printed data.

The ability to generate plots of the data presents a significant problem. First, the data is not organized in a fashion that would be well-suited to work with an existing plotting package. Although the output file can be changed to contain only numbers, there are no existing programs that could accommodate the volume of data produced by the Caveny program. One of the main considerations has to be reducing the amount of data provided to the user. The most efficient way to do this is to create a number of plots that allows the user to view a lot of data at one time. These plots are dynamic, and the variables they display and the ranges over which the data are displayed can be changed. More than one plot can be maintained at a time by the program.

An integrated environment from which the user does not need to exit offers the most significant advantages in ease of use. The goal is to provide a shell program that encourages use by those not familiar with the Caveny program or the Unix operating system. To this end, a number of features that were not critical to using the program were incorporated into the interface. A simple text editor based on the xedit sample application has been designed and included.

VI. X PLOT PROGRAM DESCRIPTION

The specifications set forth in Chapter V led to the development of a stand-alone application named xplot, a graphical user interface for the Caveny program.

A number of issues had to be resolved at the outset, since they would have significant impact on program development. The computers available in the College of Engineering at the time the project began were the IBM mainframe, Sun workstations, and a number of IBM PC compatibles. The mainframe offered no graphics capability. The PCs did not have the memory, storage, or processing capability necessary to run even the original Caveny program. The Sun workstations offered the power and graphic output capability needed. Since the operating system used on these machines was Unix, that would be the operating system for which xplot would be written.

The X Window System from the X Consortium at MIT was chosen as the graphics package for the interface program. Competitive packages available for Unix workstations that were considered include: Pixrect (Sun Microsystems' device-dependent screen drawing mechanism), PHIGS (a Programmable, Hierarchical Interactive Graphics System - designed for creating, manipulating, and rendering objects in both two- and three-dimensional space), and GL (a multi-vendor general-purpose drawing environment from Silicon Graphics). X is more appropriate for the specific application for a number of important reasons. X is widely available on most Unix platforms that support graphics output. The calls that are made to X routines are platform-independent; as such, programs written on a machine that supports X should only require recompilation to work on a different machine. Revisions of X have been available at no charge for both internal use and worldwide distribution. Most important, however, X is the only system available that provides any support for developing full-scale, interactive user applications; most of the other systems only offer libraries of drawing calls.

The choice of Unix for the operating system and X for the graphics system made C the most natural programming language to use in writing xplot. At the project's inception, C compilers were provided for use with the Sun workstations. More important, however, was the need for a language offering both data structures and dynamic memory allocation to use in conjunction with subroutine calls to X. X routines require data to be passed and returned using data structures, and they often require the calling application to both allocate and free memory. Since FORTRAN does not support either capability, C was the obvious choice.

From the outset of the program, xplot and the Caveny program were designed to work together but remain two separate processes. The Unix environment supports true preemptive multitasking, so it was possible to create an interface shell that executes the underlying solid rocket motor prediction code. From the user's standpoint, there is no need to exit from xplot. All the details of building the input file, running the Caveny program, and loading the resulting data were made transparent.

There are several reasons that support making the interface program a separate process. There is no reason why the user should have to wait for the Caveny program to finish executing before using other parts of xplot. Had the programs been integrated, there would have been no way for the user to continue building input datasets or creating plots from previous runs. The user would have had to wait for the simulation to finish before continuing the analysis. It would also have added to the complexity of the two programs if C and FORTRAN routines had been used together. The way in which arguments are passed to subroutines differs between the two languages, and special libraries that support both languages must also be used. Finally, the Caveny program could be left unmodified if two processes were used; any other implementation would have required making changes to the existing code.

The Unix operating system provides true preemptive multitasking for multiple processes on the same machine. Any number of executables can be running concurrently, and time-sharing between them is handled by the operating system. Associated with each instance of an executable is the environment under which it is run. Collectively, the executable code, its environmental variables, and the memory it has reserved for its internal data are referred to as a process.

Under the Unix operating system, there is only one way to start a new process. The fork system command creates a duplicate of the process that is currently executing, including all of its environmental variables and data. The fork command is unique in that it has a single entry point and two return points, one in the calling process and one in the process it creates. These two processes are identical, but can distinguish themselves from one another through the value returned from fork. The parent process is returned the process ID of the child process if the call succeeds or -1 if the call fails, and the child process is returned a value of zero.

The fork command only creates a duplicate of the parent process; it does not run the desired executable. For this to occur, one of the exec family of system routines must be used to change the executable associated with the child process. Note that all environment variables remain the same across a call to one of the exec functions.

The fork and exec functions are almost always used with one another. Typically, all a child process will do upon returning from a fork is transform itself into the appropriate executable using one of the exec calls. Assuming that the child process is performing some task for the parent process, the parent will often block until the child has finished executing. Under Unix, the wait command provides this facility.

The fork-exec-wait combination is used in conjunction with xplot and the Caveny code. When the user chooses to execute the Caveny code after building an input case, the interface program forks to create a clone process. The child process then immediately transforms itself to the executable name stipulated in a corresponding field in the execution dialog. This executable is named caveny by default. xplot does not wait until the child process has finished executing to restore control to the user. Note, however, that the current dataset, input files, and plots will not necessarily be updated unless specifically requested by the user. Although similar results could have been achieved using the Unix system command, system is, in general, less reliable and requires much more overhead than a fork-exec-wait combination does.

No direct communication takes place between xplot and the Caveny program. Instead, xplot is used only as a pre- and post-processing application. Input files can be generated using xplot's built-in editor. When the user selects to run the Caveny program, a file name must be specified in the execution dialog. xplot renames this file to the name of the input file expected by the Caveny code, caveny.in. The Caveny code is then run, and it generates two output files named caveny.out and caveny.plot. Either file can be viewed using the file editor dialog, and the plot file can then be loaded as the current dataset and used to create new plots. The two programs communicate through these three files only.

The organization of data generated by the Caveny program was reflected in xplot, both in the features it offered to the user and in the way it stored and manipulated the data internally. In general, the Caveny program produces two types of data: variables that are functions of both distance along the motor and time elapsed in the simulation, and variables that are functions of time alone. The model chosen for the interface had to be flexible enough to allow for both of these types of data. Instead of modifying the output file of the Caveny program, write statements were simply added at appropriate points in the code to produce a plot file with the desired format (see 2.5).

Data were stored in a three-dimensional floating point array named plotData. The X-dimension was used as discrete locations along the grain, so this dimension was limited to 30 values. The Y-dimension was used for the different variables output to the

plot file. Since the maximum number of variables that could be output was limited to 8 that were functions of both space and time and 8 that were functions of time alone, this dimension was limited to 16 values. The Z-dimension was used as an index into the time at which the variables had been written. Since this number was dependent on a number of factors (i.e., the total run duration, step size, and print interval of the simulation), the Z coordinate had to have the greatest dimension. Ultimately, this value was limited to 1024 points, a value that was generally appropriate for runs of less than one second. Figures VI.1 and VI.2 graphically show this data organization.

Time = 0.0000 seconds			
Distance	Pressure	Temperature	Mach Number
10.0000	[REDACTED]	[REDACTED]	[REDACTED]
25.0000	[REDACTED]	[REDACTED]	[REDACTED]
80.0000	[REDACTED]	[REDACTED]	[REDACTED]
Time = 0.0010 seconds			
Distance	Pressure	Temperature	Mach Number
10.0000	[REDACTED]	[REDACTED]	[REDACTED]
25.0000	[REDACTED]	[REDACTED]	[REDACTED]
80.0000	[REDACTED]	[REDACTED]	[REDACTED]
Time = 0.0020 seconds			
Distance	Pressure	Temperature	Mach Number
10.0000	[REDACTED]	[REDACTED]	[REDACTED]
25.0000	[REDACTED]	[REDACTED]	[REDACTED]
80.0000	[REDACTED]	[REDACTED]	[REDACTED]
120.0000	[REDACTED]	[REDACTED]	[REDACTED]
200.0000	[REDACTED]	[REDACTED]	[REDACTED]

Figure VI.1: Two-Dimensional Data Organization

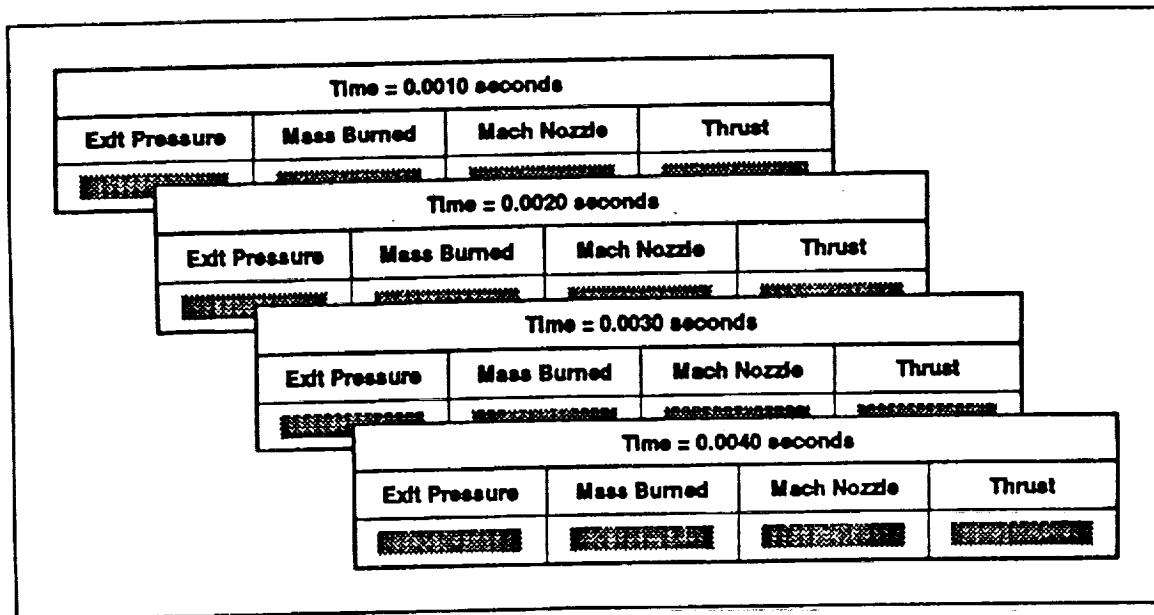


Figure VI.2: One-Dimensional Data Organization

In Figure VI.1, the two-dimensional sets of data (used for quantities such as pressure, temperature, and Mach number) have values (Y axis) for all pairs of space (X) and time (Z). The increasing values of time are shown as planes of (X, Y) pairs, which is consistent with the way the data is written to the file. In Figure VI-2, one-dimensional data (such as exit pressure, mass burned, and thrust) is shown. Here, data is a function of time (Z) alone.

This storage technique made it possible to change the way in which two-dimensional data was presented in plots. The curves could be drawn at various points in time, with the variable shown as a function of distance along the grain. However, the data "cube" could also be rotated, so the data could be drawn at various "slices" of distance, with each variable shown as a function of simulation time. Example plots that reflect this capability are given in section VII of this report.

The X Window System is a set of library subroutines that provide graphics and interface capabilities on Unix platforms. X is based on the client-server model; client applications make requests to an X server to perform certain actions, and the X server notifies the client about particular events that are of interest. X works equally well on a single computer or across a network. Each host computer can have an X server running on it, so it is possible for applications to make requests on both local

and remote hosts. X provides a platform-independent set of subroutines that client applications can use. All machine-dependent drawing and event scheduling are handled through the X server, an implementation tailored specifically for the hardware on which it is running.

X actually provides three different levels of possible interaction with client applications. At the lowest level is the X library (Xlib), a library that provides extensive routines for drawing graphics primitives (lines, rectangles, polygons, arcs, text, etc.) and basic windowing environment routines. Xlib alone is generally used only for basic drawing and as the foundation for more elaborate user interface mechanisms. The X Toolkit (Xt) is an example of one such mechanism. It provides the framework for manipulating complex graphic objects that have specific appearance and behavior called widgets. Examples of widgets include push buttons, toggle buttons, menus, scroll bars, etc. Xt is an event-driven system, in that the client application creates objects, modifies their appearance and customizes their behavior appropriate for their use. The Xt application main loop performs all polling and dispatching of events. The third level of interaction is a widget set, which greatly affects the functionality of the user interface. Short of writing custom widgets, the things that can be done in a program are limited by the functionality of the widgets employed. This three-tiered organization of interface code is shown in Figure VI.3.

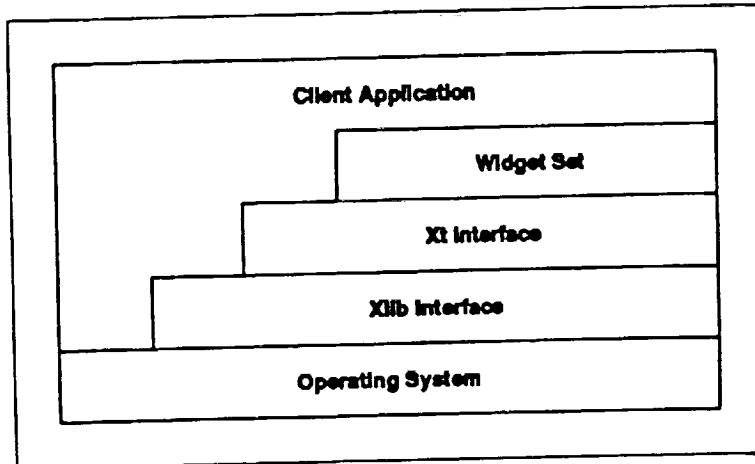


Figure VI.3: X Toolkit Application Design Model

The client application can interact with any or all levels of interface provided: it can change the value, appearance, or behavior of one or more widgets through the widget set, control the overall application through Xt, issue drawing requests in a window through Xlib, or perform any operating-system specific calls concurrently.

The xplot interface program uses all three levels of X. The widget set chosen was the Athena widgets provided as examples with the distribution from the X Consortium. Although Athena widgets do not have as nice an appearance or quite the functionality of commercial widget sets, they are adequate for the somewhat limited needs of the program. Further discussion of the widgets used in xplot and Xt in general is given below.

As previously stated, the X environment is event-driven. Rather than performing certain actions in a specified order, as is customary in traditional programs, X applications must respond to events, which are usually user inputs to the program, either directly or indirectly. For example, at the Xlib level, a plot must be redrawn not only when created but also whenever it is exposed (from underneath other windows) or resized. All of the techniques discussed for drawing plots really refers to the actions taken whenever one of them must be updated.

A simple layout was chosen for the plots generated by xplot. Only simple, rectangular 2D scatter or line plots are provided. Many simplifying assumptions about the kind of data being displayed can be made since the application is designed specifically to work with the output of the Caveny program. The independent variables are either axial distance along the grain or simulation time. The number of variable data traces displayed are limited to sixteen for time-dependent and eight for distance-dependent plots. The general layout characteristic of these plots is shown in Figure VI.4.

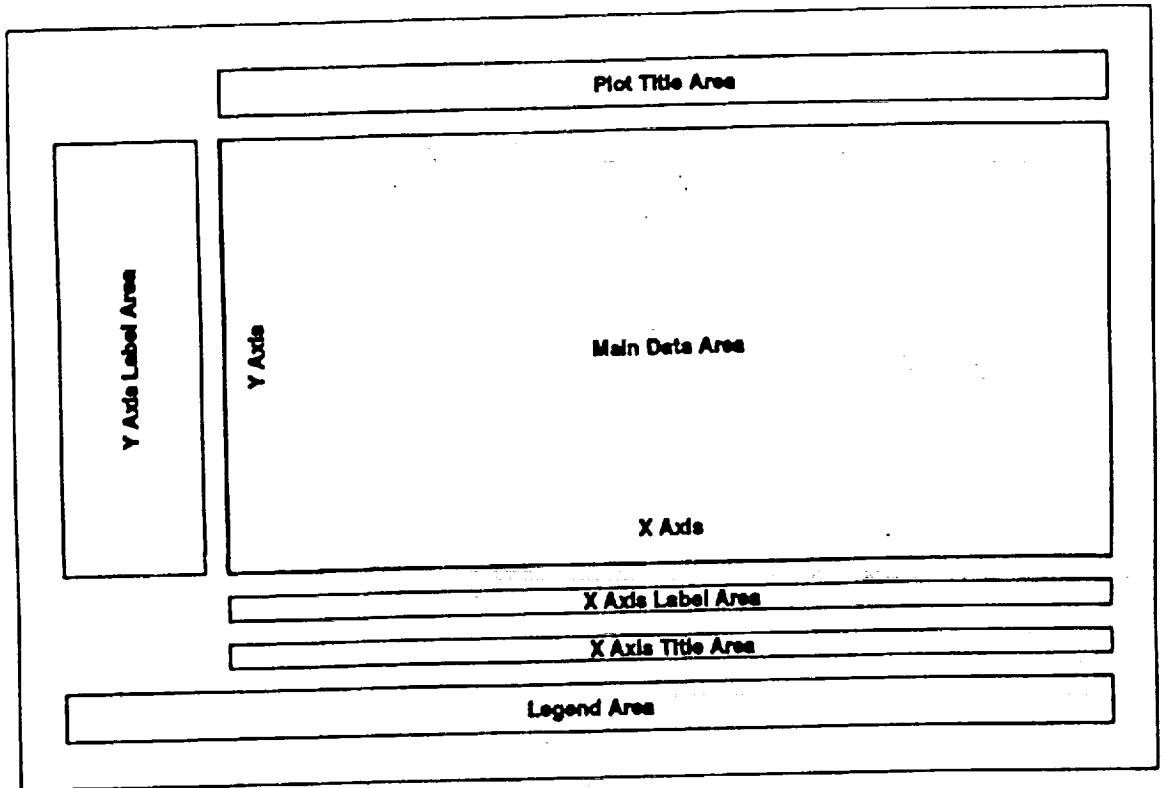


Figure VI.4: Plot Layout Design

The size of a plot depends on both the size of the window enclosing it and the borders chosen for it. Only the data curves are drawn in the data area; all other plot components are drawn in the borders around it. The width and height of the plot then can be expressed as:

$$W_{plot} = W_{window} - B_{left} - B_{right}$$

$$H_{plot} = H_{window} - B_{top} - B_{bottom}$$

The elements of the plot are drawn in the following order: plot title, plot legend, x-axis title, axis labels, axes, gridlines, and then the plot data. This order was chosen to insure that various elements of the plot neither obscure nor erase the others.

The plot title is centered over the main plot area. The font used for the title is Helvetica 18 bold. The width of the text string is computed using the XStringWidth function. Once the

width of the text string is known, the x-component of the title can be calculated using the following equation:

$$x_{title} = B_{left} + (W_{plot} - W_{title}) / 2$$

The plot legend is placed across the bottom of the plot area. Each element of the legend is drawn as a small rectangle having the same color as the data it represents, followed by a string label corresponding to the same data. Up to sixteen sets of data can be shown in the plot legend area, with each line containing as many as four identifying markers.

The plot labels are drawn using successive calls to XDrawString. The sprintf function from the C standard library was used to print the text to a string in the format chosen. These axis labels are drawn at regular intervals along the appropriate axis.

An axis title is drawn for the x-axis only, since X does not support drawing text rotated to arbitrary angles. The x-axis title consists of the independent variable, either axial location or time, drawn in Helvetica 10 Bold, centered below the main plot area. It also indicates which slice of either time or space is currently being displayed.

The plot axes are drawn using a 1 point black line along the bottom and left edges of the plot area. Since all of the data that could be plotted, including dependent and independent data, contained only positive values, no provision was made for drawing the lines where $Y = 0$ or $X = 0$ inside the plot area.

The grids on the plot are drawn at regular intervals corresponding to the number of axis labels. These lines are drawn as 0 width lines inside the bounding rectangle that formed the main plot area.

The plot data is the most complicated aspect of the plot to draw. Each trace of data has its own attributes, and there can be up to 16 traces of data on a single plot. The location of any single data point can be calculated from interpolation:

$$x_{data}[i] = B_{left} + (X[i] - X_{min}) / X_{range} * W_{plot}$$
$$y_{data}[i] = B_{bottom} - (Y[i] - Y_{min}) / Y_{range} * H_{plot}$$

Using these equations, data can lie outside the plot area rectangle if the scales have been manually set. Such values of data are either less than the minimum or greater than the maximum values of the scales. To insure that the data drawn in the main data area of the plot frame does not exceed the limits of the

area, the calls for drawing data were clipped to the plot data area. The clip mask of the current graphics context is set to the rectangle R(Bleft, Btop, Wplot, Hplot). Explicitly setting the clip mask overrides the mask that is set by an expose event. Rather than have the program maintain a list of all areas exposed, combine them with the plot area into a single clip mask, and then redraw the plot, xplot simply redraws the entire plot on any expose event.

Xlib provides line attributes as part of the graphics context, or current drawing environment. These attributes include foreground and background drawing colors, line width, style (solid, on-off dashed, double-dashed), join mode (rounded, mitered, or beveled), and end cap style (butt or rounded). Implementing the various types of lines that can be specified in xplot only requires calling XSetLineAttributes with the appropriate values before any drawing commands.

Since X is implemented using client-server methodology, the speed with which it can perform drawing operations is as limited by the rate it can receive commands as it is by the speed of the processor. To reduce the amount of communication (often network) traffic, Xlib provides routines for drawing arrays of objects as well as the routines used to draw single objects. Points, lines, line segments, rectangles, and arcs, can all be drawn either individually or in groups. To maximize refresh rate, xplot assembles a table of all the points for a given trace of data, then calls XDrawLines.

Descriptions of the Athena widgets used in creating the xplot application are given in Table VI-1.

Table VI-1: Widgets Used in xplot

Class	Description	Use
Box	Geometry management of arbitrary widgets in a box of specified dimension.	Most dialogs, to organize buttons and text fields into logical groupings.
Command	A rectangular button used to invoke an action.	In all dialogs to perform any necessary actions.
Dialog	A window, used to hold other widgets, that can be placed and iconified by the window manager.	All pop-up selection panels are instanced from dialogs.
Form	Organizes the layout and size of its child widgets.	Organizing sub-objects in all dialogs.
Label	A rectangular area containing descriptive text.	All dialogs, to indicate the function of an area or an associated text field or button.
List	An array of text entries, drawn in a column, from which one or more can be selected.	Scrolling lists, such as the command and history summary and the input file variables.
MenuButton	A rectangular button from which a pull-down menu can be displayed.	Only used in the Top Level menu to contain all menus.
Text	An area containing text that can be either display-only or editable.	All dialogs, whenever the user is prompted for data entry; in dialogs to display large amounts of read-only text.
Toggle	A button that can be in one of two states, either on or off.	Setting options about the appearance of plots.
Viewport	An area with scroll bars that allows the user to view an object larger than the area provided.	Scrolling lists, such as the command and history summary and the input file variables.

The dialogs used in xplot are all constructed using similar techniques. A shell widget interacts with the window manager, allows a window to be moved, resized, restacked, or iconified, and can contain only a single child widget. A dialog shell widget was used for all of the dialogs other than the Top Level. A form widget was created as the shell's only child, and it handles geometry management of all the child widgets. Normally, these children were arranged by function into related groups.

The way the Format dialog is constructed is representative of the dialogs used in xplot. A diagram of this dialog and its components is given as Figure VI-5. A single Form widget is the only child of a dialog shell. A number of box widgets are its children, arranged in an order that logically divides functions of the dialogs. These boxes are attached to one another using

the geometry management facilities provided by the form widget, so the dialog is only as large as it needs to be to contain all of its children. The text fields and command buttons inside the boxes are organized in columns to minimize the space they occupy.

Only a fraction of the widgets created for an Xt application need to be modified during program execution. The values of some of these widgets will be modified by the user, and some will be changed by the application to reflect the current state of the data it maintains. In general, widgets that need to be modified include text fields (used for data entry and presentation), command buttons (used to reflect currently available or acceptable choices), and toggle buttons (used to reflect a current state of a selection).

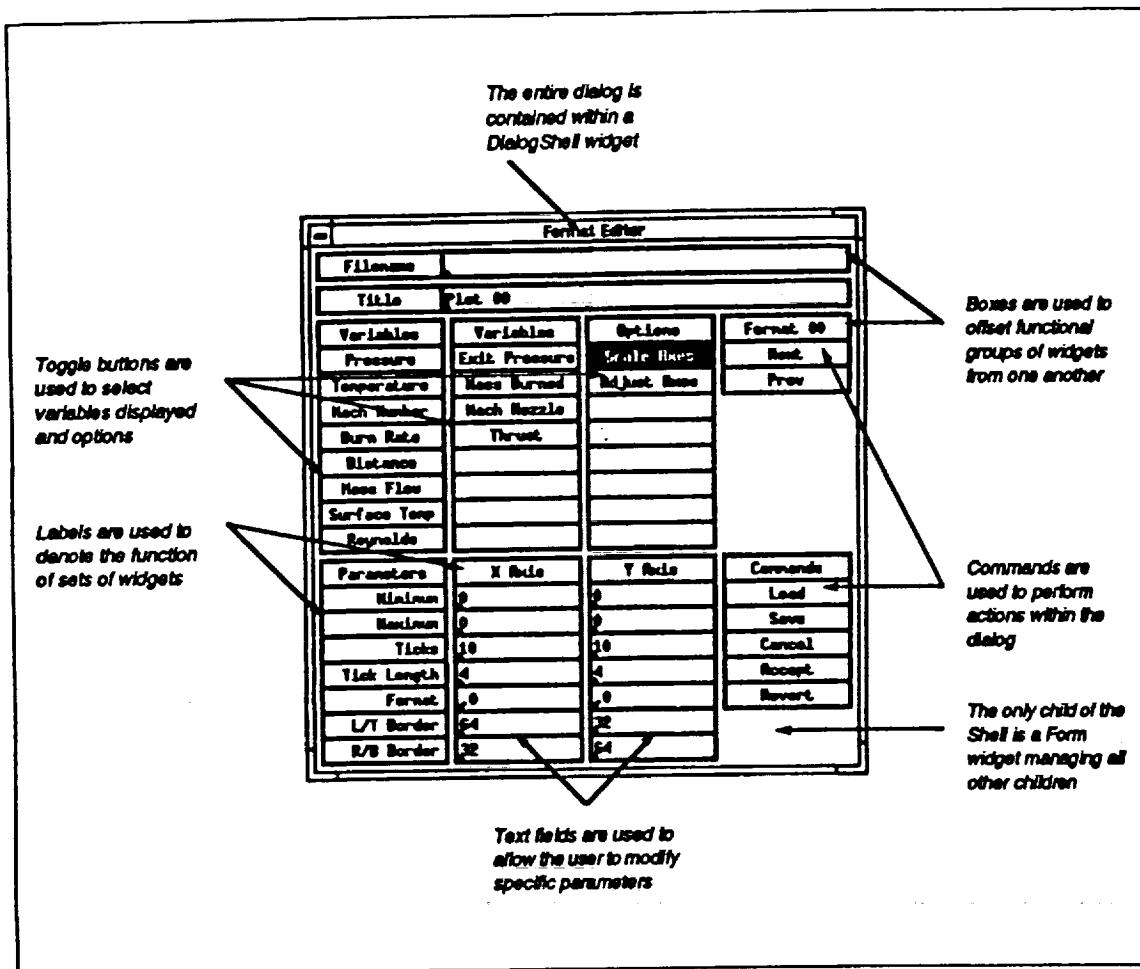


Figure VI.5: Example dialog design

In an Xt application, the ID associated with a particular widget is returned by the function used to create it. For most of the widgets used in xplot, that ID is retained only as long as needed. For instance, all widget creation routines require the widget's parent to be passed as one of the arguments. However, for decorative and geometry-management widgets (like form, box, or label), once their children have been created, there is no reason to maintain their ID, since it is not necessary to modify them. For the widgets that did need to have their values read and set, the variable-arguments form of the Xt get and set functions, XtVaGetValues and XtVaSetValues, are used.

Xt provides higher-level event processing than is possible through Xlib alone. Instead of notifying the client application that the user has pressed and released one of the mouse buttons at a certain location on the screen, it might instead notify that a button has been selected, a menu has popped up, or a scroll bar had been paged. Xt provides these higher-level events through a mechanism it defines as callback routines. Callback routines tell a widget how to respond to certain well-defined events. Most events will not be of interest to the application, and the widget's default behavior will be the only action that is either necessary or appropriate.

In addition to the graphical interface provided by xplot, a command-line language was written. The grammar supported by xplot is very small. Since there are less than fifty possible legal tokens in the vocabulary, there is no need for a separate lexical analyzer or parser (such as those provided by the Unix utility programs lex and yacc). Instead, each line is simply broken down into its constituent components by a single function call. This function uses string comparison routines to arrive at allowable input. To keep the code as straightforward as possible, the actions and options that can be set from the command-line are mapped to equivalent functions already present from within the graphical interface.

The source code for xplot is organized into files, grouping subroutines that deal with related aspects of the application. This organization is used for two main reasons. First, to encapsulate the data and routines into a similar area, so that as little global data and global subroutine calls are necessary. Second, to make compilation both more efficient and quicker during development. The source code files correspond to the various main aspects of the program, i.e. the command-line interface, the internal data, the plot formats, drawing the plots, etc.

A complete listing of the source files used to build xplot and their associated functions is given in Tables VI-2 and VI-3. All source files have a .c suffix, and all header files have a .h suffix.

Table VI-2: xplot Source Files

Filename	Lines	Description
xpcommands.c	248	Passes input from the command-line, translates commands into their graphic equivalents.
xpdata.c	250	Manipulates all of the internal data.
xpformats.c	358	Loads and saves formats from files.
xpgeneral.c	137	Creates graphic contexts, loads colors and fonts.
xpgeometry.c	169	Responds to all configure-notify and expose events generated by the window manager.
xpinput.c	716	Loads all input datasets.
xplot.c	69	Initializes all of the Xt interface, calls all the subroutines to create all of the widgets, then calls <code>XtAppMainLoop</code> .
xpmenus.c	582	Creates menus in the Top-Level dialog and responds to commands issued from these menus.
xpplots.c	731	Draws all of the aspects of the plots.
xpwidgets.c	907	Creates all widgets in the application.

Table VI-3: xplot Header Files

Filename	Lines
xpcommands.h	13
xpdata.h	25
xpformats.h	13
xpgeneral.h	23
xpgeometry.h	8
xpinput.h	9
xpmenus.h	14
xpplots.h	23
xpwidgets.h	47

VII. XPLOT USER INSTRUCTIONS

The xplot program is designed to be usable both by those familiar with graphical interfaces as well as those more comfortable with command-line interfaces. Although it is graphical in design, xplot does provide rudimentary support for a command-line language. A discussion of the graphical interface and the commands are presented first, then each of the text commands are discussed in terms of their graphic equivalents.

The top-level dialog of xplot, the window that is displayed when the program is first run, contains controls to access all capabilities of the program. This dialog is shown in Figure VII.1.

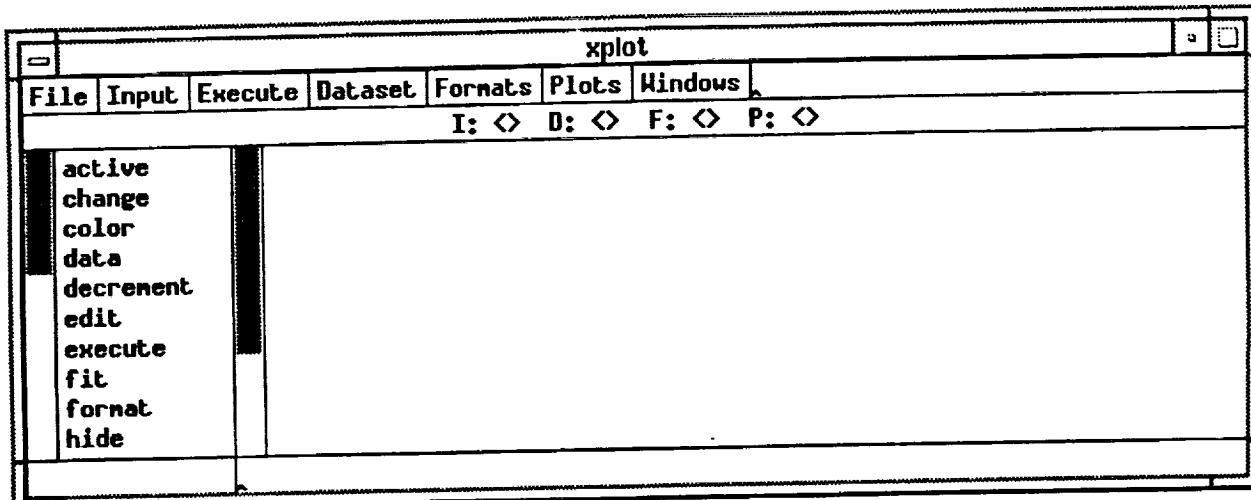


Figure VII.1: xplot Top Level Dialog

A menu bar, located across the top of the dialog, provides menus for performing certain functions. To the immediate right of the dialog is an area for entering the pathnames associated with certain files recognizable by xplot, such as datasets, plot format files, etc. Below the menu bar are three areas used in conjunction with the command-line interface. To the left is a scrolling list containing all command verbs that xplot recognizes. To the right is a scrolling list containing the last fifteen commands entered by the user. Below these two scrolling lists is the text field in which all commands must be entered.

Selecting either the verbs from the command list or previous commands from the history list will place them in the command text field, but it will not execute the command. To execute a command, the user must press either return or enter in the text field.

The top-level menus "File", "Input", "Execute", "Dataset", "Format", "Plot", and "Windows" represent the logical divisions of the program. The "File" menu allows the user to select files for viewing, save the current file, or quit the application. The "Input", "Dataset", and "Format" menus are identical in layout, but each deals with a different kind of file that can be read by xplot.

The Input File dialog allows the user to create and modify input files for the Caveny program without actually editing the text files it uses for input. This dialog is shown in Figure VII.2.

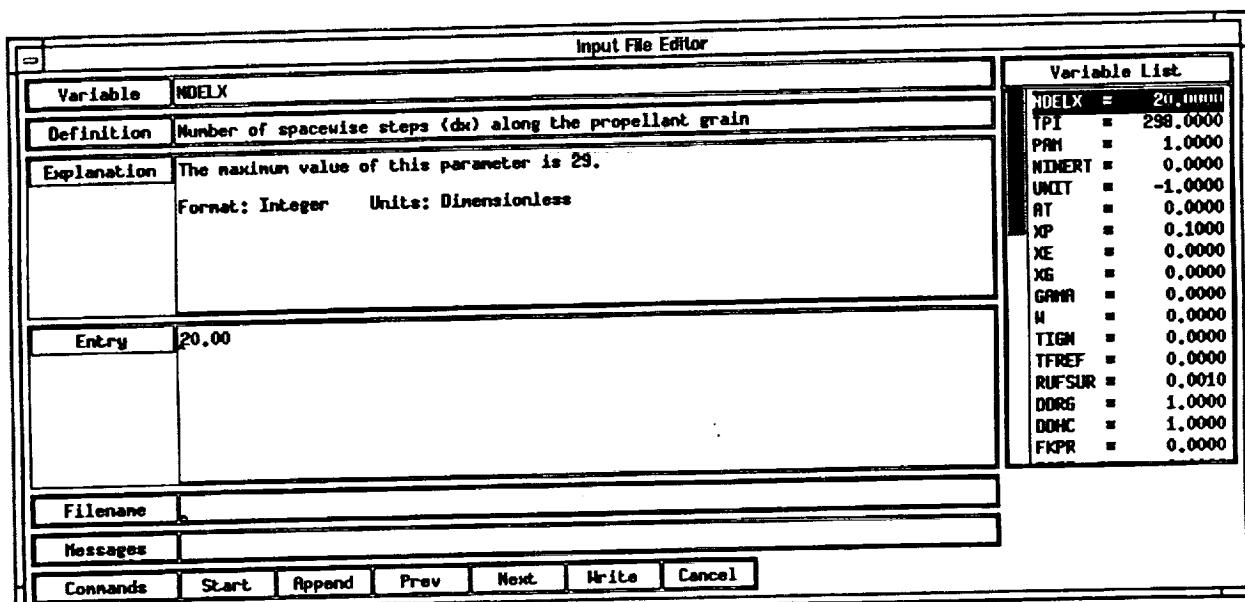


Figure VII.2: xplot Input File Editor Dialog

On the right side of the dialog a scrolling list denoted Variable List shows all of the case description variables and their current values. If the user selects one of these list elements, the four fields of the dialog described below will be updated to reflect the properties associated with this variable.

Once the user has selected a variable to edit, either through the scrolling list or with one of the commands located across the bottom of the dialog, the top four text fields will be filled in with the current data about that variable. The FORTRAN name of the input parameter will be displayed in the Variable field. A brief (one-line) description of the parameter's significance to the Caveny program will be visible in the Definition field. The Explanation section will contain up to six lines of pertinent information about the variable. This information will usually consist of the internal representation of the variable (i.e., INTEGER, REAL, etc.), the FORTRAN format in which the variable will be read (i.e., F10.4), the units that it should be expressed in terms of, limits on the values it can take on, etc. All three of these sections are read-only.

The contents of the Entry section can vary greatly depending on the variable selected. For simple scalar quantities, which are summarized in the Variable List, a single value will be displayed.

The Filename field is a read-write text field that contains the pathname to be used in all read/write operations. When the dialog is first raised, this field will be empty. At this point, the user can either start a new input case file or enter a valid pathname in this field. The value provided in this field will be used for all read and write operations. As such it is possible for a user to load in a new input dataset without saving the changes made, save the new values in place of the old file, or save the current values to a new filename. The messages text field will contain pertinent information about the operations that the dialog performs and diagnostics about errors that it may encounter in attempting to carry out the user commands.

The commands available in the Input File dialog are summarized in Table VII.1. For more information about the format of the Caveny program input file and the variables defined in namelist NAME, see Section IV.

Table VII.1: Input File Editor Dialog Commands

Command	Action
Start	Returns to the first variable available and selects the first variable name in the scrolling list.
Prev Next	Advances to the previous/next variable available and highlights the previous/next variable displayed in the scrolling list. When either of these commands are selected, the value in the Entry field will be substituted for the previous value of the current variable, and this value will be reflected in the Variable List.
Write	Saves the current values to the filename specified in the Filename text field.
Cancel	Dismisses this dialog, but maintains all of the current settings.

The Execution dialog contains text fields that allow a user to configure the specifics of running the Caveny program from within xplot. This dialog is shown in Figure VII.3.

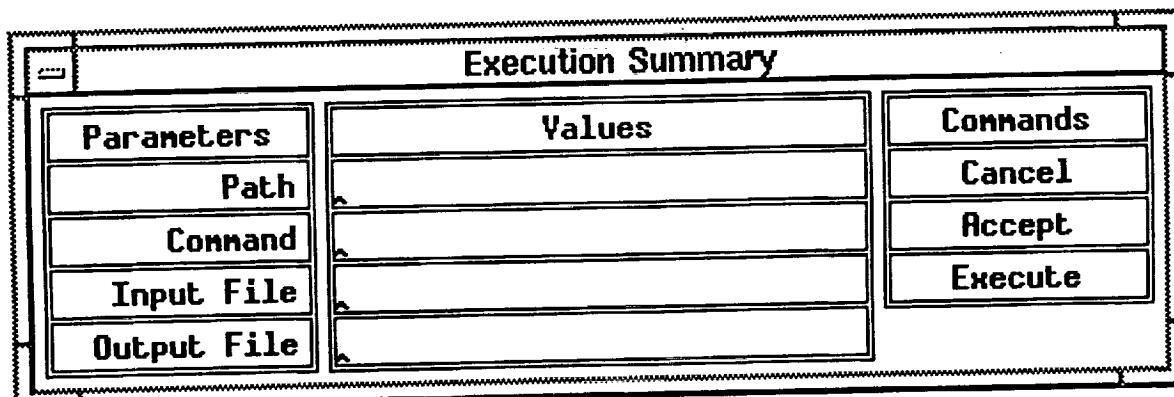


Figure VII.3: xplot Execution Setup Dialog

The Execution dialog consists of four text fields. The Path field displays the pathname, either global or local, that is the directory that will be searched for the Caveny program. The Command field displays the name of the Caveny program executable. By default, the path is simply the local path "./" and the executable is named "caveny". The Input File and Output File fields name, respectively, the files expected by the Caveny program as input and the filename that will be used for its output. By changing these filenames, various input cases and output datasets can be maintained by the user. The user can also run the Caveny program from this dialog, once he has selected the proper configuration, using the Execute button. The commands available with the Execution dialog are summarized in Table 14.

Table VII.2: Execution Dialog Commands

Command	Action
Cancel	Dismisses this dialog, saving all of its current settings.
Accept	Dismisses this dialog, saving all of its current settings to internal values for these parameters.
Execute	Immediately runs the Caveny program using the parameters currently specified in the dialog; saves all current values.

The Plot Manager dialog contains controls that allow the user to configure up to eight plots simultaneously. This dialog is shown in Figure VII.4.

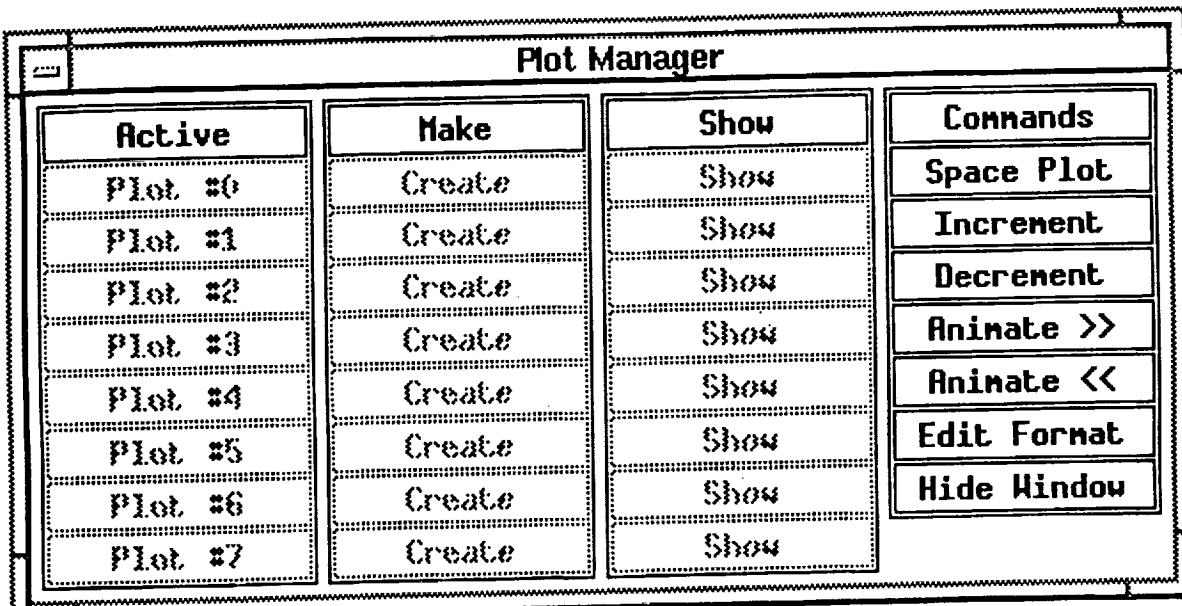


Figure VII.4: Plot Manager Dialog

The Commands column allows the user to switch the independent variable, move either along the grain or in time, or animate any of the eight available plots. The commands available in the Plot Manager dialog are summarized in Table VII.3.

Table VII.3: Plot Manager Dialog Commands

Command	Action
Space Plot/ Time Plot	Changes the orientation of the currently active plot. Toggles between the plot being displayed as variables as functions of time or axial distance.
Increment Decrement	Changes the view to the next/previous time or spatial plane location.
Animate >> Animate <<	Quickly increments/decrements the view to the beginning/end of the time or spatial stations.
Edit Format	Brings up the Format dialog, with the values shown corresponding to the currently active plot. If the format dialog is already visible, then change the dialog so that it displays the values associated with the currently active plot.
Cancel	Dismisses this dialog, saving all of its current settings.

The Format Dialog allows the user to edit the format used to display any of the eight plots available. This dialog is shown in Figure VII.5.

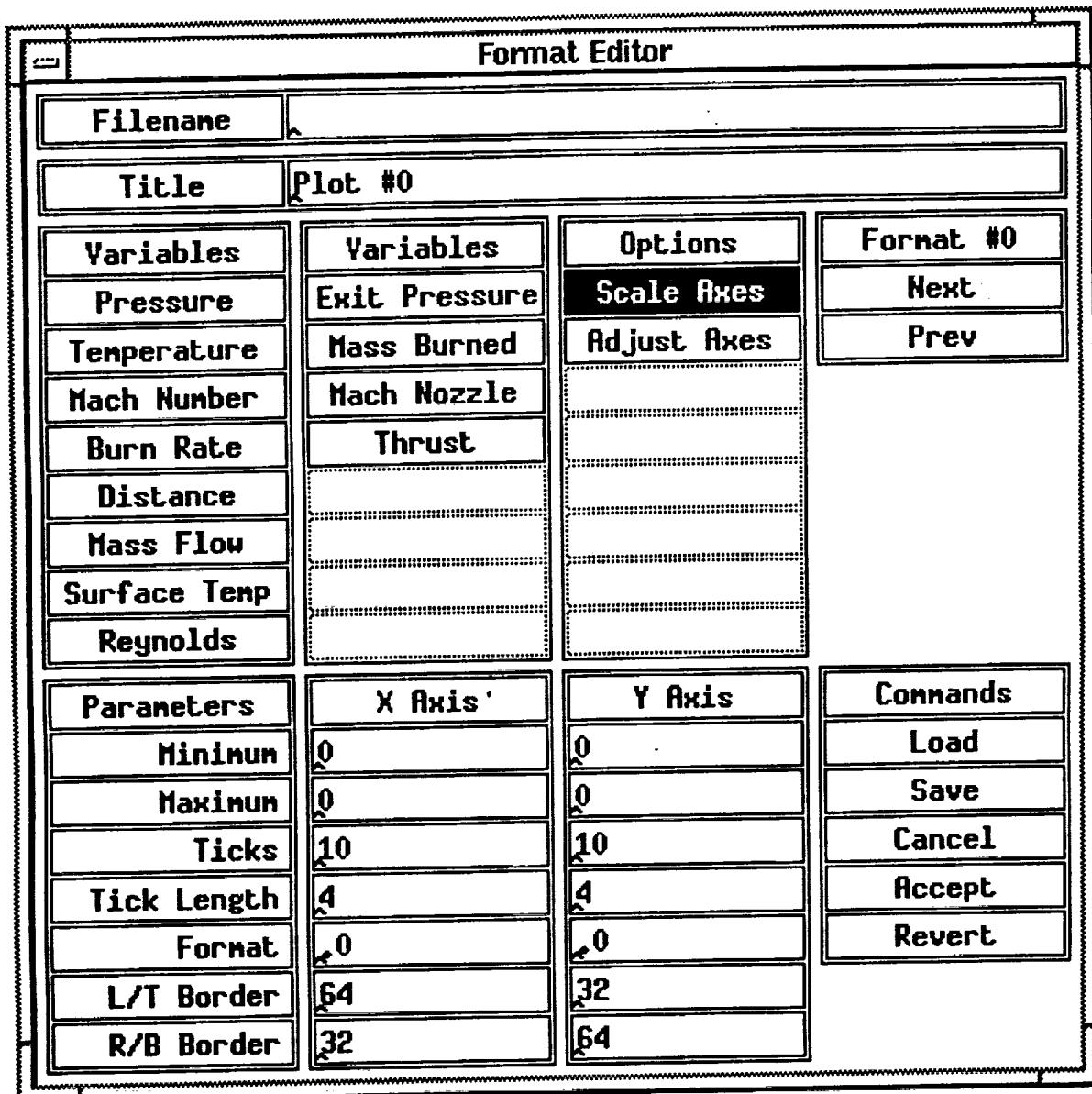


Figure VII.5: xplot Format Dialog

The format dialog will allow the user to edit the formats of plots other than the active plot, and these plots need not be either created or visible to edit their format. These formats can be both loaded from and saved to external files for future use. This dialog works in conjunction with the Data Attributes dialog to modify the internal state of the formats of the plots.

Formats can be saved or loaded just like datasets. When the program is begun, a default set of format values will be installed for each available plot. The user can edit the format to the style preferred, then save and recall these settings using the Filename field. The Title field specifies the label that will appear centered across the top of the plot.

There are two columns of toggle buttons labeled Variables. The left column contains all of the variables in the current dataset that are functions of both position and time. The variables listed in the right column are functions of time alone. Note that nothing will be displayed on the plot if the user selects only time-dependent data from the Format dialog and attempts to create a space plot. The organization of data in a plot file is discussed in Section VI.

By selecting one of the toggle buttons in the Options column, the user can determine which features to use on the current plot. The first option, "Scale Data", forces the Y axis to take on minimum and maximum values corresponding to the minimum and maximum values of the data displayed on the plot. The second option, "Smooth Data", sets the minimum and maximum values and the number of gridlines to even numbers that are more appealing. None of the other six options are currently defined.

Values for both the x- and y-axes can be set using the text fields at the bottom of the dialog. If the "Scale Data" option is not active, then the minimum and maximum values of either axis can be set manually. Note, however, that both minimum and maximum must be set for each if any are to be set. The number of gridlines and labels for either axis can be set in the same way. The minimum number of gridlines is 1, and the maximum is 20. The borders affect the way the plot is sized and drawn. The text fields in the x-axis area control the size of the left and right borders, and the text fields in the y-axis area control the bottom and top borders. All plot decorations appear outside the main plot area, so the borders may have to be resized to give the plot an appropriate appearance.

The Format field dictates the number of decimal points and total field width that is used in displaying the axis labels. These values should be specified in terms of standard C formatting patterns as used with the standard input/output

function sprintf . This format is very similar to the syntax used for REAL variables in FORTRAN FORMAT specifications. The commands available in the Format Dialog are summarized in Table VII.4.

Table VII.4: Format Dialog Commands

Command	Action
Prev Next	Switches to the previous or next format specification. Any of the formats numbered [0-7] can be edited at any time.
Load Save	Loads or saves the particular format being displayed to or from the filename currently in the Filename field.
Cancel	Dismisses the Format dialog. Retains the current contents of the dialog, but does not save those changes to the plot or to a file.
Accept	Changes the properties of the plot corresponding to the current format to the values input.
Revert	Changes the values in the dialog to either those values last saved or the default values. This function is essentially the same as switching to either the previous or next format and then switching back.

The Data Attributes dialog allows the user to customize the appearance of the data displayed in the main content area of a plot. This dialog is shown in Figure VII.6.

Data Attributes Editor				
Data	Colors	Dashes	Colors	Dashes
Pressure	3			
Temperature	3			
Mach Number	3			
Burn Rate	3			
Distance	4			
Mass Flow	5			
Surface Temp	6			
Reynolds	7			
Exit Pressure	8			
Mass Burned	9			
Mach Nozzle	3			
Thrust	3			
	12			
	13			
	14			
	15			

Figure VII.6: xplot DATA Attributes Dialog

The first column contains an index into the colors specified in the colors files. The color of the label containing the variable name will be changed to this color whenever the dialog is updated. The number of dashes can be specified in the second column. The format for entries in this column is the number of pixels that should be drawn "on" followed by the number of pixels that should be drawn "off". For instance, 20 pixels on followed by 10 pixels off would be entered as "20 10". There are no commands associated with the Data Attributes dialog, since it only reflects additional information about the format currently selected with the Format dialog.

The File Edit dialog provides an editable text field in which the user can load, view, make changes to, and save ASCII text files. This dialog is shown in Figure VII.7.

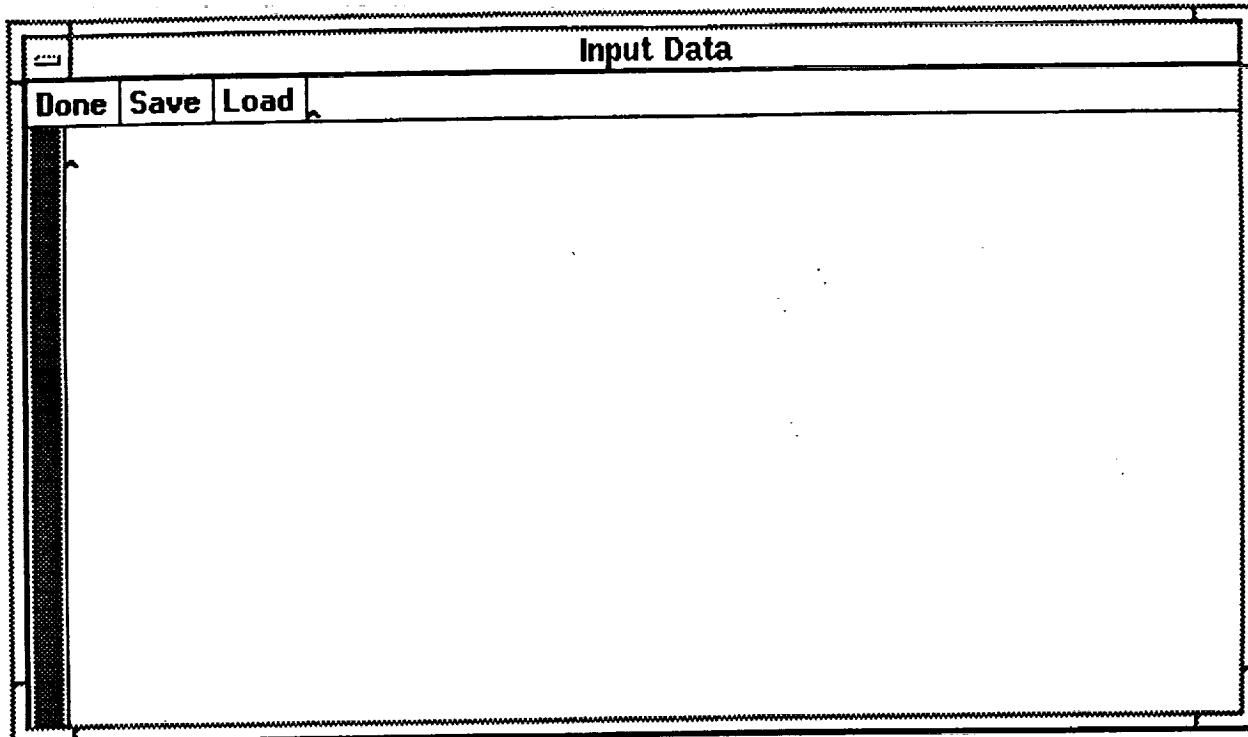


Figure VII.7: xplot File Editor Dialog

The dialog is modeled after the xedit application included in the standard distribution of the X Windows from MIT. This rather simple dialog contains three action buttons, a single-line text field for entering filenames, and a large text field for displaying the contents of text files. This dialog is included to provide the user a simple capability of editing text files from within the application, without having to quit and restart the interface program or edit field in another shell tool. The load and save buttons read in or write out their contents based on the filename specified in the single-line text field. The contents either of the text buffer or the external text file will be overwritten upon executing one of these commands. The commands available in the File Edit dialog are summarized in Table VII.5.

Table VII.5: File Editor Dialog Commands

Command	Action
Done	Dismisses the dialog. Maintains the current state of the text displayed, but does not save those changes to a file.
Save	Saves the current state of the text to the filename given in the Filename text field.
Load	Loads the contents of the filename specified in the Filename field into the current text buffer. Overwrites the contents of the text buffer with the contents of the file specified in the Filename field. If the file does not exist (i.e., if the filename represents a new file), then that file is opened (if possible) and the contents of the text buffer are cleared.

The View Dataset dialog provides a simple read-only text field area displaying the currently loaded dataset. This dialog is shown in Figure VII.8.

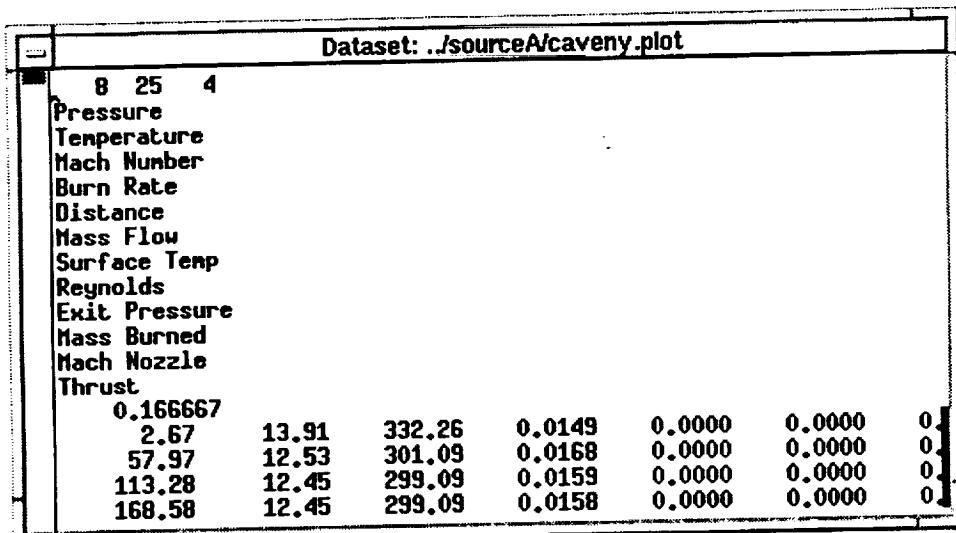


Figure VII.8: xplot View Dataset Dialog

This dialog is provided only to allow the user to easily view the values as data in the plots created. This dialog functions just like the File Edit dialog, except it always contains the text of the current dataset, and the text in it can

be neither edited nor saved. The dialog can be resized according to the width of data it displays. There are no commands associated with the View Dataset dialog. It must be shown or hidden from either the Dataset menu or the Window Manager dialog.

A simple command-line interface (or CLI) is provided for those users more comfortable with the more conventional means of entering commands. Note that each of the commands available through the CLI corresponds to one of the commands available in the graphical interface. A summary of the commands available is given in Table VII.7. In Table VII.7, the notation \$1 refers to the first argument, \$2 to the second, etc.

Table VII.7: Command-Line Interface Verbs

Command	Args	Description
active	plot	Changes the active plot to the one specified in \$1.
color	data color	Changes the color associated with the data specified in \$1 to the color specified in \$2. Equivalent to changing the appropriate "color" field on the Line Format dialog.
data	file	Loads the file specified in \$1 as the current dataset.
decrement (dec)	plot	Decrement the plot specified in \$1. If no argument is given, the active plot is decremented. Equivalent to selecting "Decrement" in the Plot Manager dialog.
edit	file	Loads the file specified in \$1 as the text to be edited in the File Edit dialog and raises this dialog. If no file is given, then the previous filename is used.
execute (exec)	path exec	Executes the Cavy program from the path specified in \$1 and the application name specified in \$2. If neither argument is given, the defaults are taken from the values most recently specified in the Execute dialog.
format	format	Changes the format being edited in the Format dialog to the value specified in \$1.
hide	plot	Hides the plot specified in \$1. Equivalent to selecting "Hide" for the appropriate plot in the Plot Manager dialog.
increment (inc)	plot	Increments the plot specified in \$1. If no argument is given, the active plot is incremented. Equivalent to selecting "Increment" in the Plot Manager dialog.
input	-	Loads the file specified in \$1 as the input dataset to be used in the Input File Editor dialog and raises this Dialog. If no file is specified, then the dialog is raised with a new input dataset.
load	path	Loads a dataset from the full pathname given in \$1. If no argument is given, it attempts to re-load the dataset most recently used with a "load" command from the "Dataset" menu.
plane	-	Sets a time/space plane on the currently active plot to \$1. If the value specified in \$1 is less than one or more than the time/space planes available, no action is taken.
plot	-	Creates the plot specified in \$1.
quit	-	Terminates execution of xplot.
save	format	Saves the current format specified in \$1.
show	plot	Shows the plot specified in \$1. Equivalent to selecting "Show" for the appropriate plot in the Plot Manager dialog.
space	plot	Toggles the plot specified in \$1 to being a space plot. If no argument is given, the active plot is toggled. Equivalent to toggling the "Space plot/Time Plot" button on the Plot Manager Dialog.
time	plot	Toggles the plot specified in \$1 to being a time plot. If no argument is given, the active plot is toggled. Equivalent to toggling the "Space plot/Time Plot" button on the Plot Manager Dialog.

The xplot application has a number of external files that must be present to run properly. In addition, a number of additional files can be used to configure the application according to the user's preferences, or to provide additional support for interactions with other programs.

In each of the following example listings, a vertical bar and all text following it should be interpreted as comments and should not be inserted into the input files.

XPlot.clr. This file specifies the colors available to be used in the line format dialog. The file is composed of a number of lines containing a number and a valid color name as specified in the X11R4 color database. Blank lines in the input file are ignored. These number-color pairs allow the user to define the colors that will be available for the various traces of data. An example file excerpt is given as Table VII.8:

Table VII.8: Xplot.clr. File Format

1	White		Color #1 = White (255, 255, 255)
2	Black		Color #2 = Black (0, 0, 0)
...			
18	Red		Color #18= Red (255, 0, 0)
19	Green		Color #19= Red (0, 255, 0)
20	Blue		Color #20= Red (0, 0, 255)
...			

The XPlot.var file contains entries for all of the data to be placed in a Cavy program input dataset generated by the xplot program. Each variable entry in this file consists of a number of parameters surrounded by a leading and a trailing double-quote. The separate fields are terminated by a trailing slash (/). The fields of a variable entry are (in order): the variable list index, the FORTRAN name of the variable, A short definition of the variable, its default value (in appropriate units), and an optional long description. An example format of the entries is given in Table VII.9.

Table VII.9: XPlot.var File Format

"		Leading double-quote
0/		Variable list index
NDELX/		FORTRAN variable name
Number of spacewise steps(dx)		Short variable description
along the propellant grain/		...
20/		Default value
Maximum value = 29		Long variable description
		...
Format: Integer Units: None		...
"		Trailing double-quote

The application-defaults file. The resources specified in the XPlot file located in the same directory with the xplot executable will be loaded in as fallback resources. Since user-specified resources take precedence over fallback resources specified by the application, resources set in either the user's .Xdefaults file or an XPlot file in the user's home account will override these settings.

Plot format files. Unlike the input files already described, there can be any number of plot format files, and they can have any name. The default format file, which is read upon program start-up, is named XPlot.def. This file is a simple text file enumerating the options selected in a plot format dialog. Each of the fields stored in the dialog is reflected in the file, in the same order as they appear in the dialog. Although there is no reason why the user could not edit this file using either the Input File dialog or a separate text processor, there is no reason to do so. These files are read and created by the Plot Format dialog directly. A sample listing for the XPlot.def file is given as Table VII.10.

Table VII.10: Sample XPlot.def File

Plot #0	Plot title
0 0 0 0 0 0 0	Space data shown
0 0 0 0 0 0 0	Time data shown
0 0 0 0 0 0 0	Options selected
64 54 32 32	Border (L, T, R, B)
10.4 10.4	Axis label formats
10 10	Number of axis gridlines/labels

The example plots shown in this section were all produced by xplot with data generated by the Caveny program. All example plots used the default values for format attributes from the XPlot.clr and Xplot.def files. The setup required for the plots is described in the text following the plot. The data for these plots was generated by the Caveny code from the sample input file given as Appendix A.

Figures VII.9-VII.12 show the development of pressure in the motor at four different times (30, 60, 90, and 120 milliseconds, respectively). These plots trace the propagation of the initial igniter motor pressure wave down the length of the grain. Figures VII.11 and VII.12 are shown to the same scale to illustrate the effects of combustion on the chamber pressure. At 90 milliseconds, the grain has just begun to burn, and the spike towards the aft end of the section results from the igniter mass flow. At 120 milliseconds, much more of the grain has begun to burn, and combustion gases from the head end have made their way down the chamber.

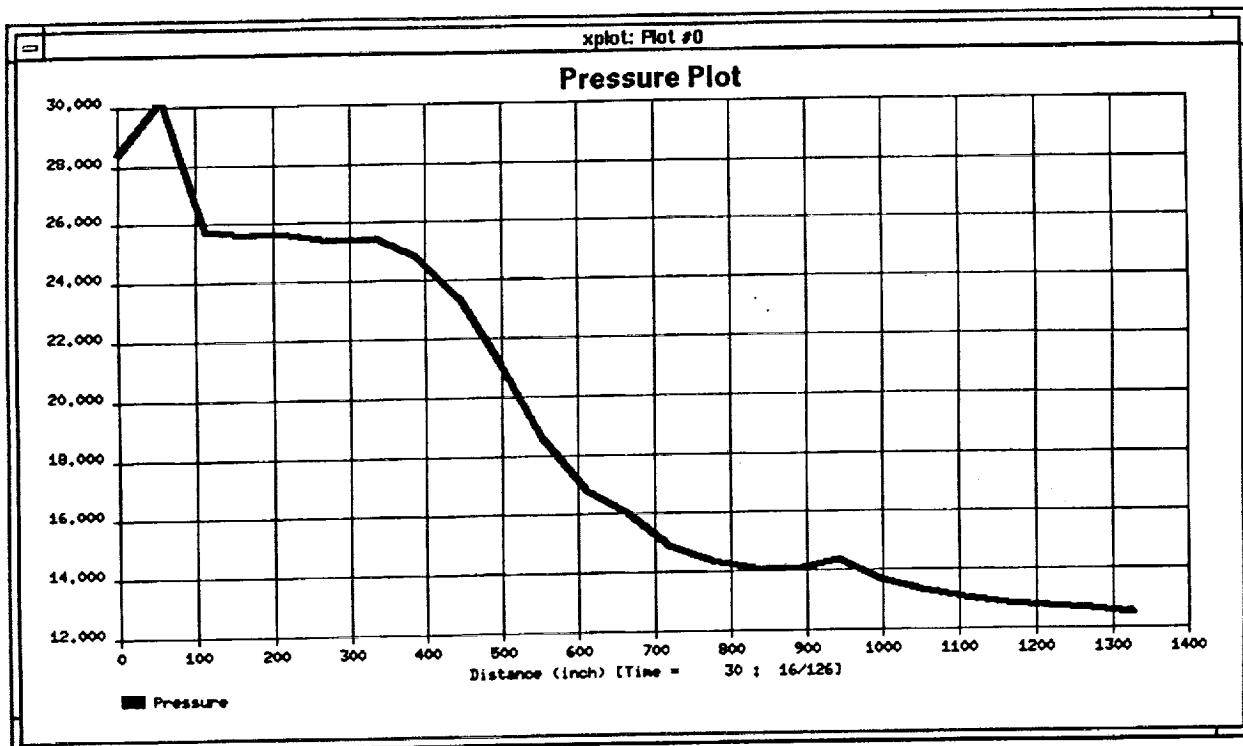


Figure VII.9: Pressure versus Location at Time=30 ms

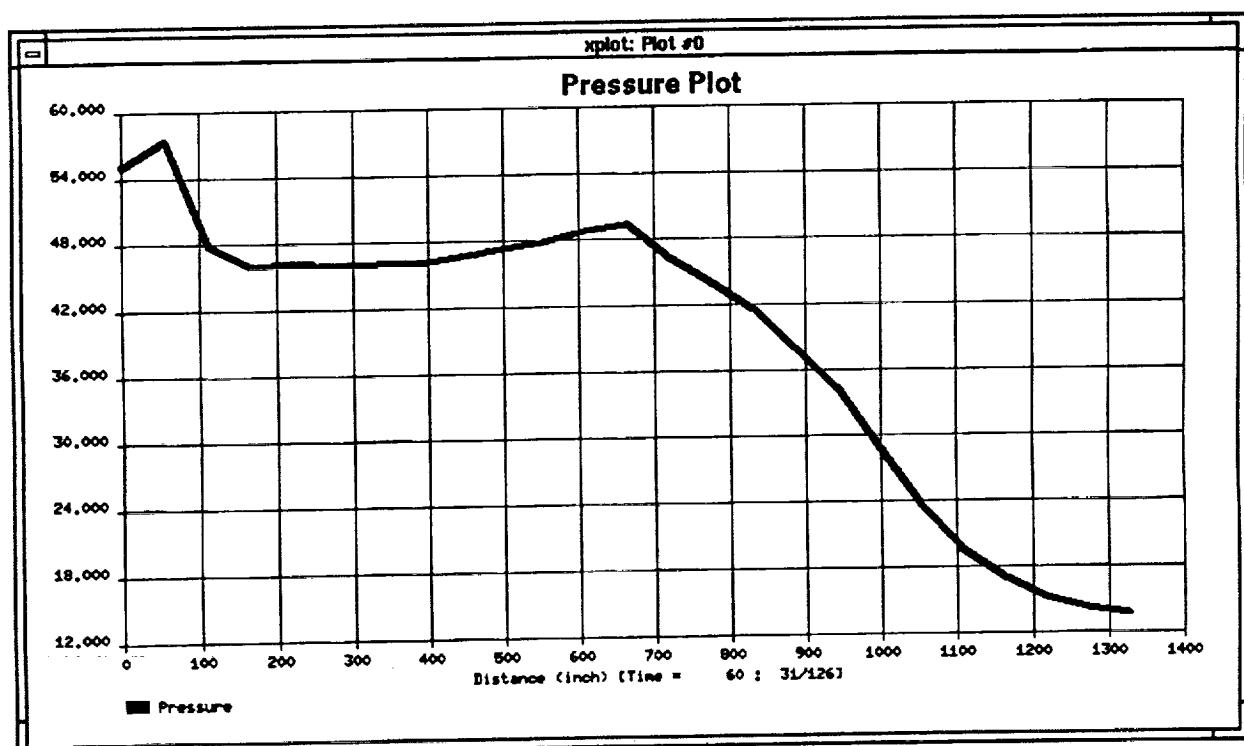


Figure VII.10: Pressure versus Location at Time=60 ms

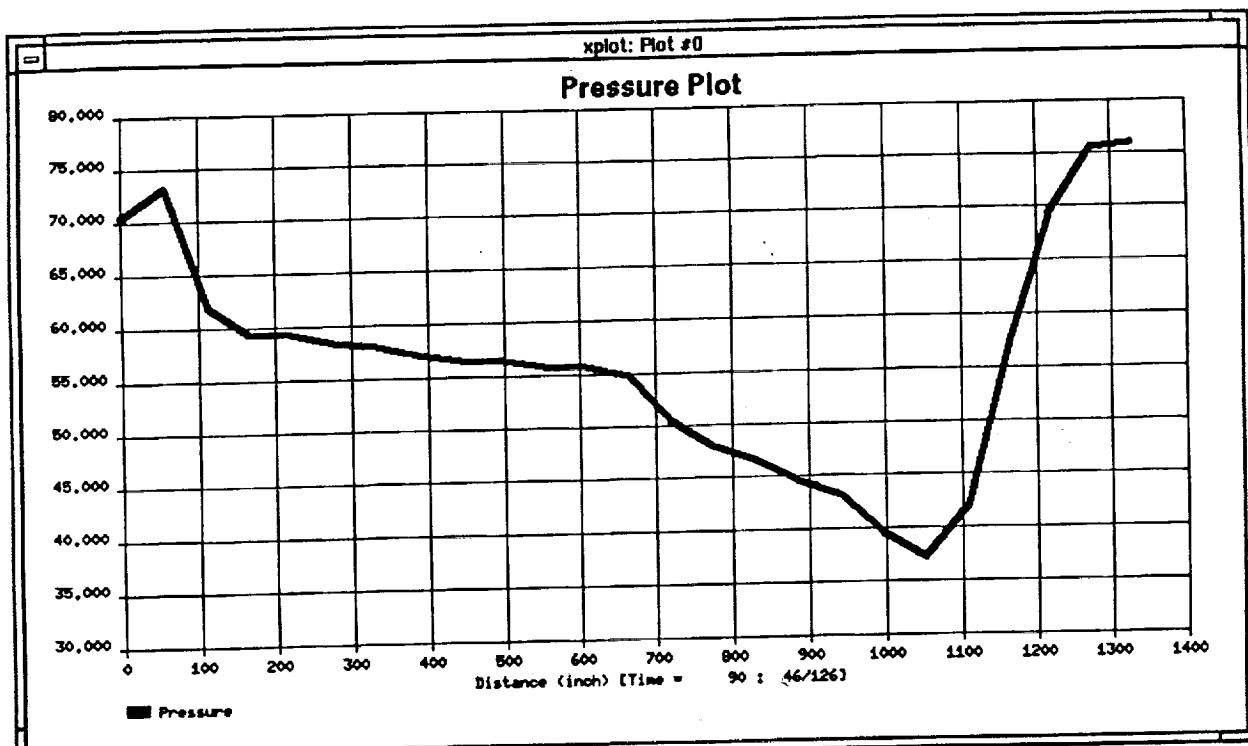


Figure VII.11: Pressure versus Location at Time=90 ms

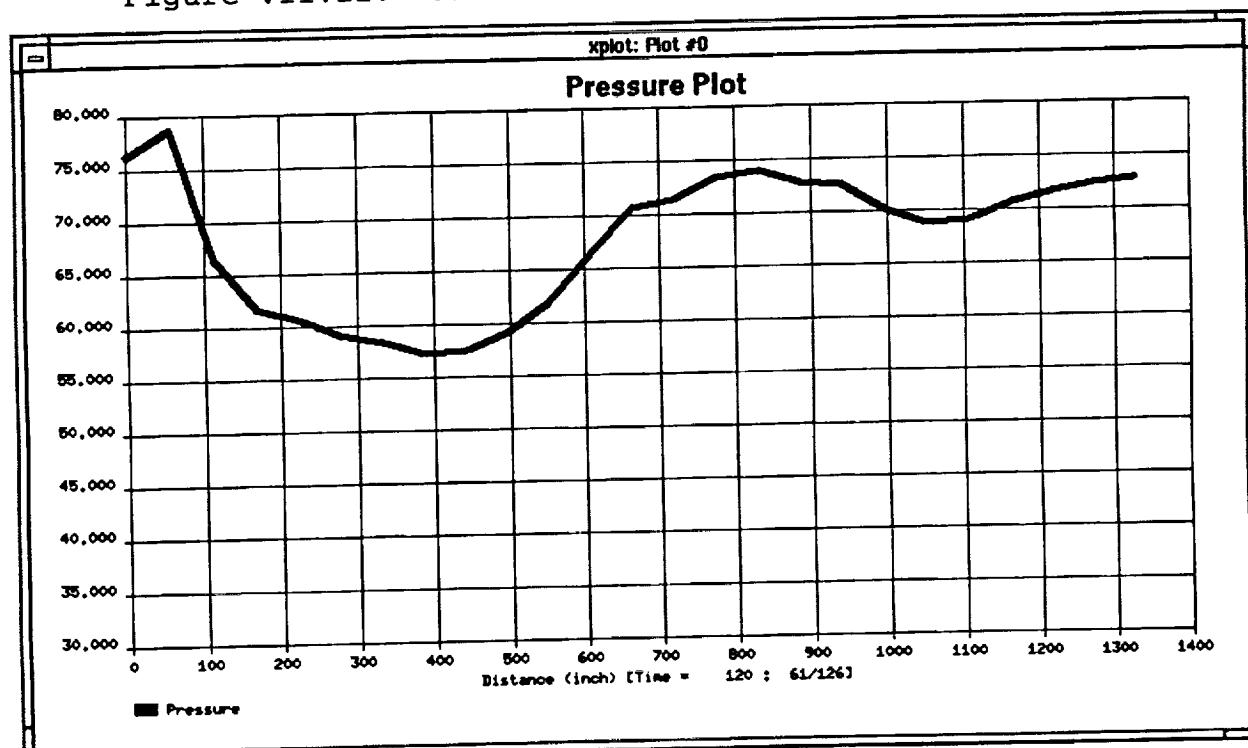


Figure VII.12: Pressure versus Location at Time=120 ms

These four plots are typical of the default type produced by xplot. All of the data are presented as a function of distance along the motor geometry. Figures VII.9 - VII.12 represent four time slices of the entire motor. Each plot contains a maximum of 30 points per curve, since this number is the maximum that the Caveny program allows for axial stations. Pressure was the only variable selected from the format dialog. The time was set using the Increment and Decrement buttons on the Plot Manager dialog. The scaling and number of gridlines were set manually. The number of decimal places for the x-axis were left at their default value of zero, but the y-axis labels were set to a 0.3 format.

Figures VII.13 and VII.14 show the data generated by the Caveny program in a different fashion. These plots illustrate the time history of a scalar quantity at two different axial distances along the motor. In this particular example, the chamber pressure is displayed as a function of time at the head end and aft end stations. Each of these plots contains 126 data points, since the Caveny input file was configured using its print interval and total simulation time to generate this number. This viewing option is set using the Space Plot/Time Plot toggle button on the Plot Manager dialog. With this option, the user can quickly switch back and forth in the manner chosen to view the data. The station is selected using the Increment and Decrement buttons on the same dialog. The scales, grids, and title were each set manually.

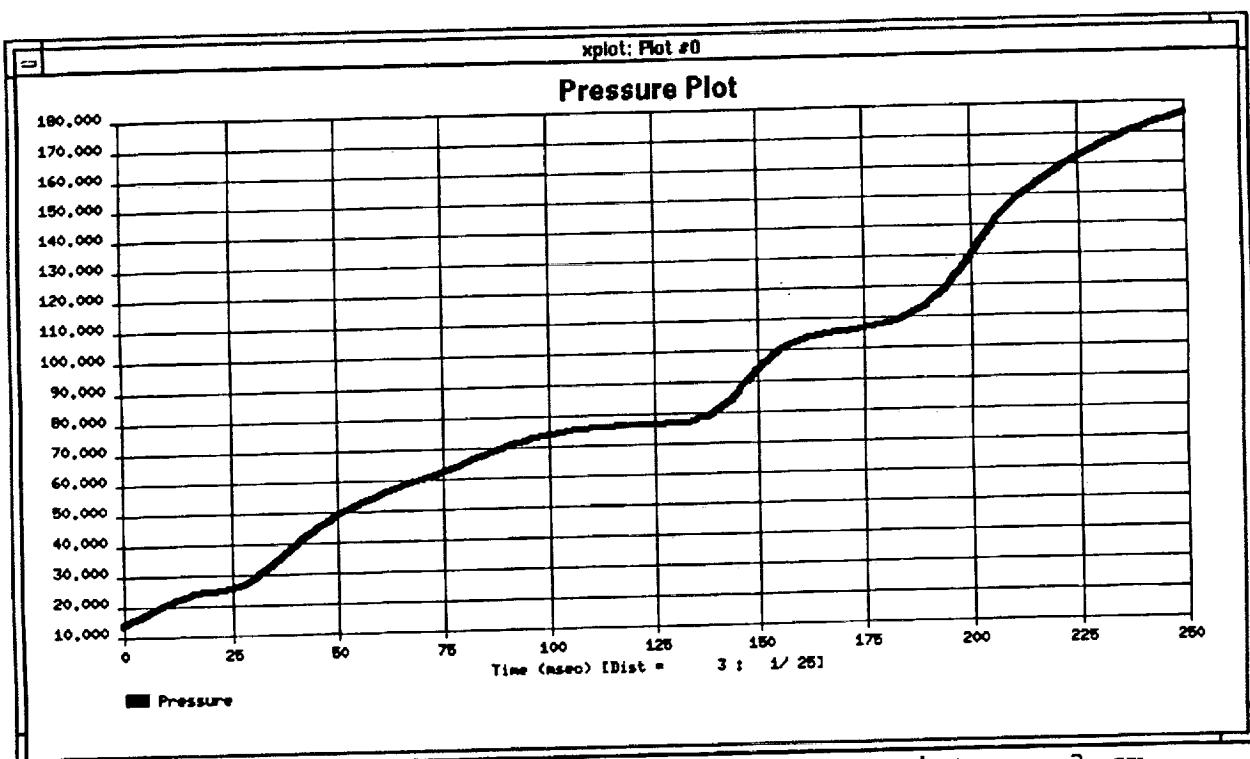


Figure VII.13: Pressure versus Time at Distance=3 cm

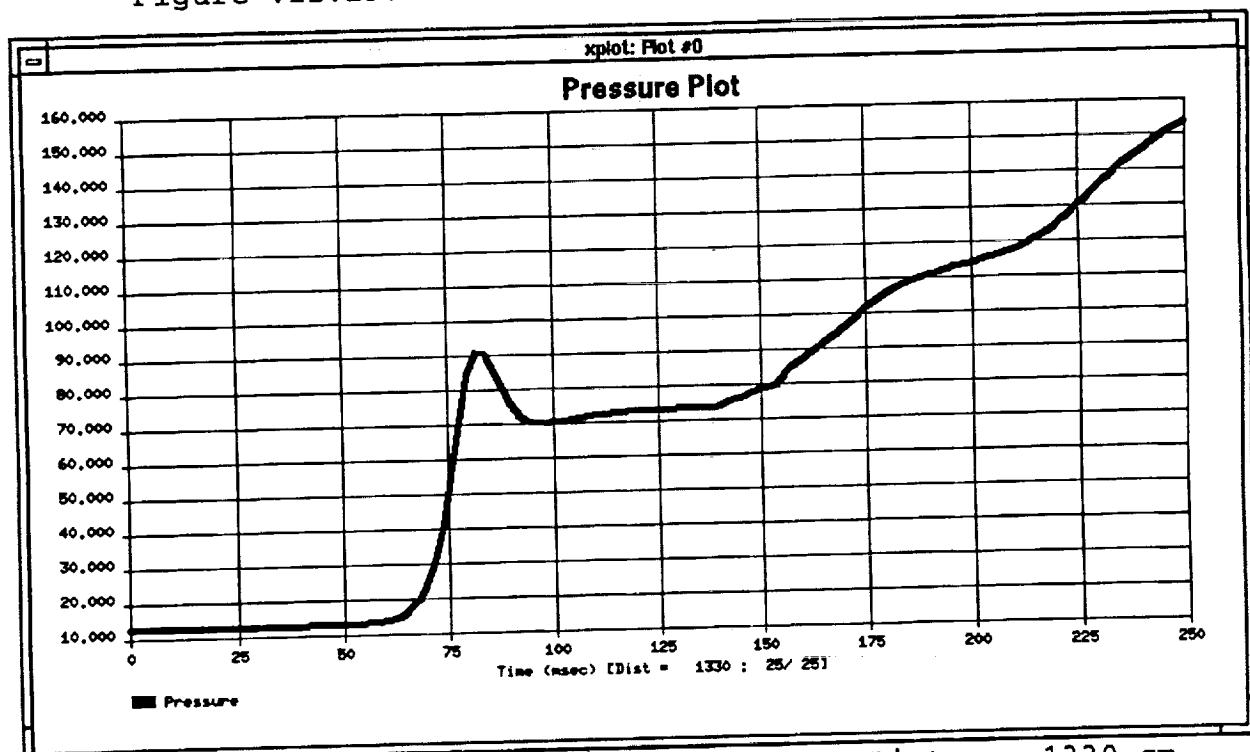


Figure VII.14: Pressure versus Time at Distance=1330 cm

Figures VII.15 and VII.16 show that multiple traces of data can be displayed on a single plot. In these two figures, the Mach number and grain burn rate are shown as functions of distance at two different times in the simulation (90 and 180 milliseconds). Of interest in the first plot is the spike in Mach number and the sharp drop-off in burn rate near the aft end of the motor. The spike again corresponds to the igniter discharge, and the burn rate reflects the fact that the aft end of the motor has not yet begun to burn. By 180 milliseconds, both curves have flattened out to more stable values all along the motor. In each plot, the legend reflects the colors associated with the traces for the two variables.

To configure these two plots, Mach number and burn rate were the only variables selected using the Format dialog. To make the traces print more distinctly, the colors and dashes for burn rate were changed from their default values. Using the Format Lines dialog, the burn rate color was set to black, and the dashes were set to 20 on 10 off (by entering "20 10" in the variable's appropriate field). In these two figures, the plots were manually titled and automatically scaled.

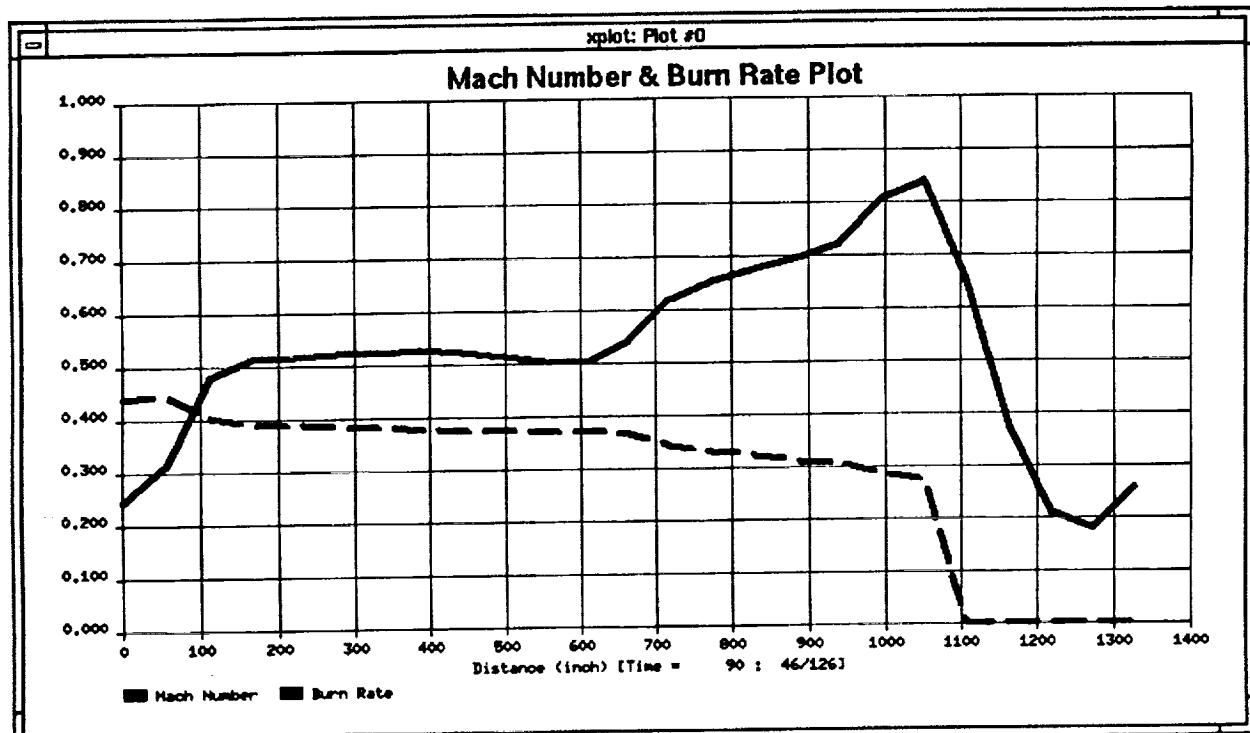


Figure VII.15: Mach Number & Burn Rate versus Distance at Time=90 ms

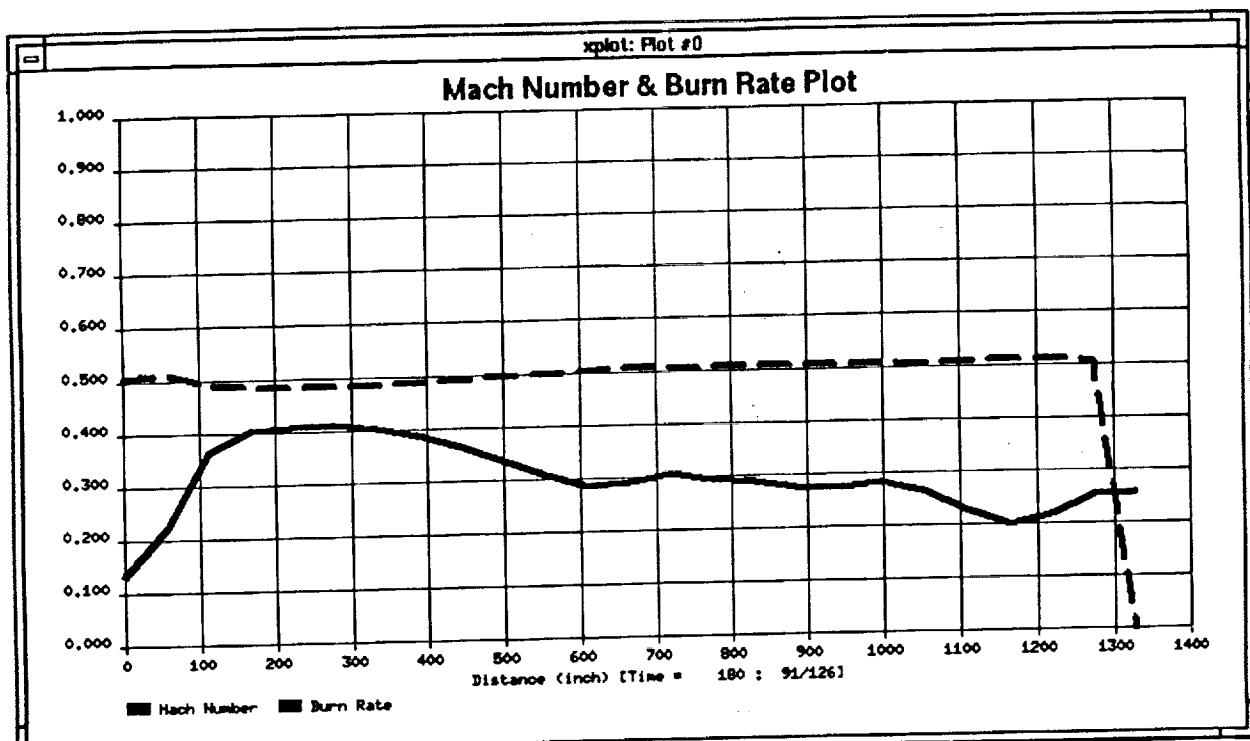


Figure VII.16: Mach Number & Burn Rate versus Distance at Time=180 ms

Figures VII.17 and VII.18 are presented as examples of variables that are functions only of time and not distance along the grain. In these two figures, the Reynolds number at the exit of the motor and the thrust generated by the motor are plotted. The curve for the thrust produced is in accordance with the plot of aft end pressure shown in Figure VII.14. The thrust is zero until the igniter gases begin to be discharged at approximately 65 milliseconds. The Reynolds number at exit is a predominantly linear function with the exception of the ignition spike from 60 to 100 milliseconds.

To create plots of variables that are time-dependent only, the Time Plot option must be selected from the Plot Manager dialog. If this option is not chosen, no data will be drawn in the plot frame. Note that the Increment and Decrement buttons in this dialog do not function if only time-dependent variables are selected. In all other aspects, these variables are configured and displayed in exactly the same way as the other variables when plotted as functions of time.

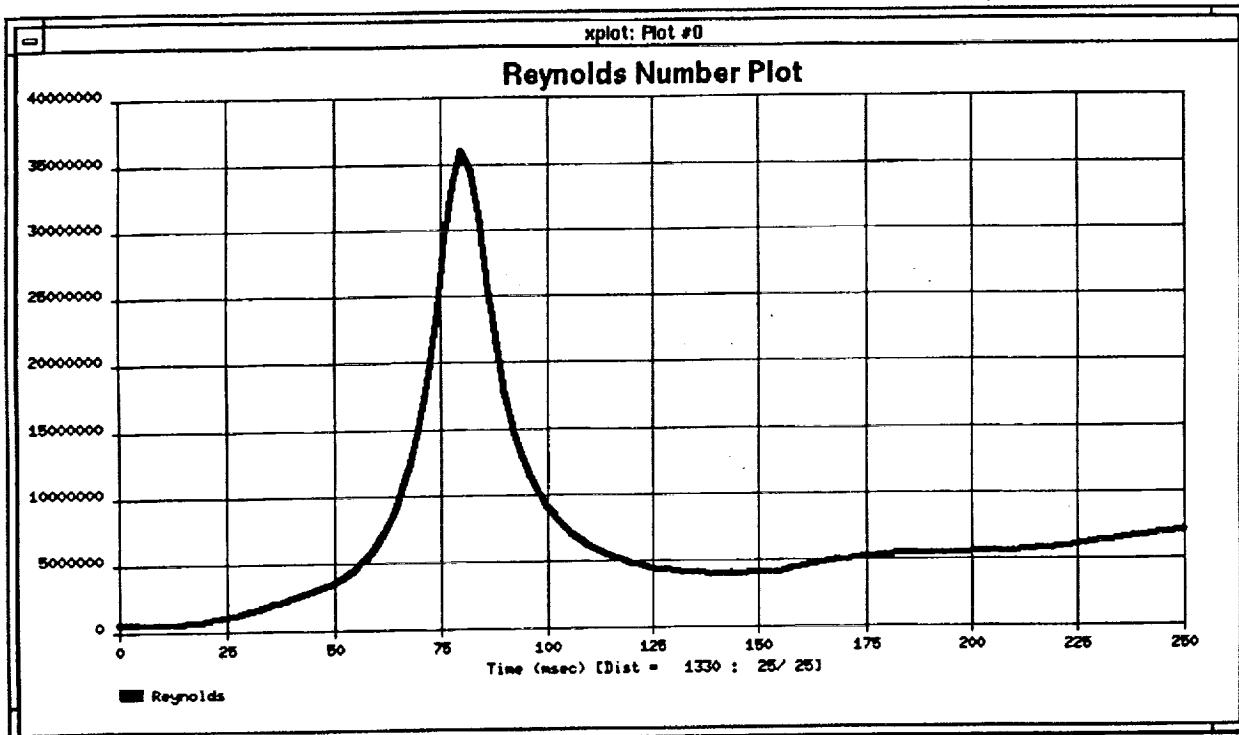


Figure VII.17: Reynolds Number at Exit versus Time

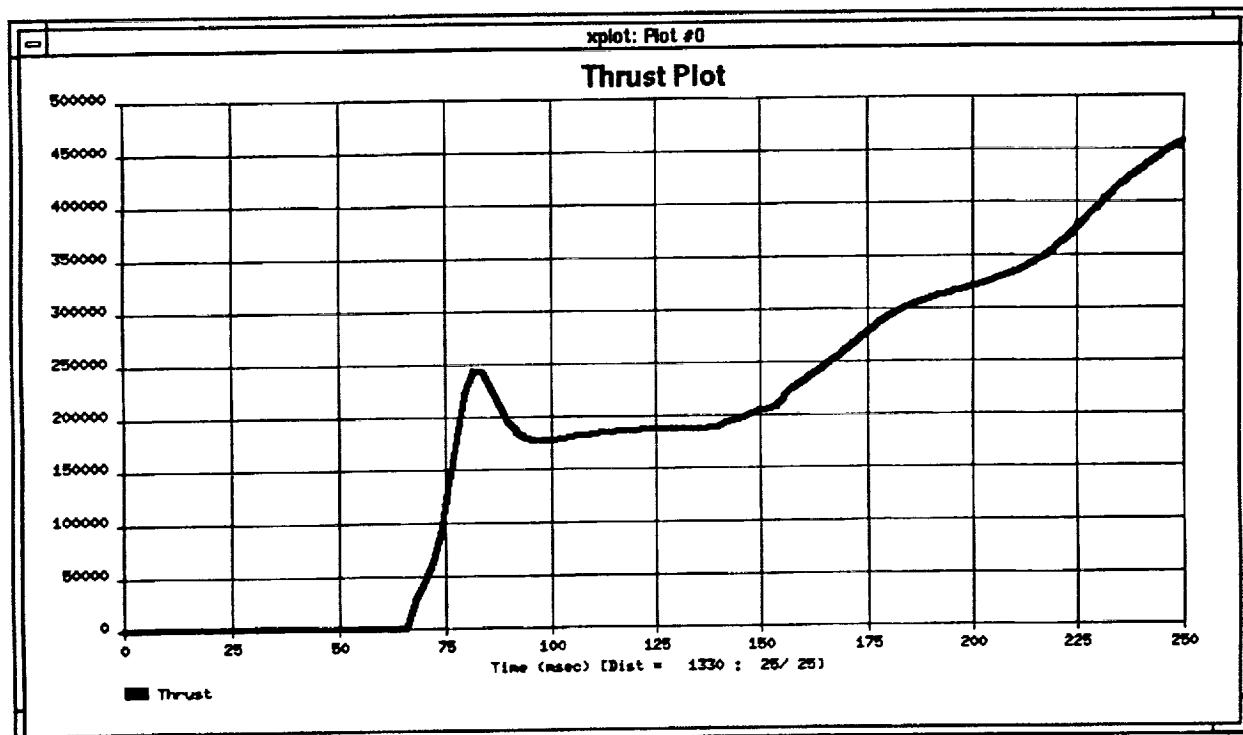


Figure VII.18: Motor Thrust versus Time

APPENDIX A

Sample Caveny Program Input File

```

1      MM-1 17DEC77 FLAME SPREAD CONTROL AT HEAD END ,DYNAMIC THRUST CHECKOUT
2      &NAME
3          TMAX = 0.002,
4          TPRINT = 0.001,
5          DELTAT = 0.00016,
6          NDELT = 24,
7              TPI = 298.89,
8              PAM = 0.84717,
9              UNIT = -2.0,
10             TIGN = 2685.4,
11             TFREF = 3353.0,
12             W = 28.18,
13             GAMA = 1.136,
14             AT = 2325.81,
15             XP = 2.67,
16             XE = 1329.972,
17             RUFSUR = 0.01,
18             FKPR = 0.0011,
19             ROPR = 1.758,
20             CPR = 0.3,
21             SIGP = 0.0009939,
22             RREF = 1.054,
23             PREF = 68.08,
24             EBC = 0.0,
25             BREXP = 0.33,
26             DE = 145.64,
27             CM = 0.97844,
28             ALFAD = 12.31,
29             GAMAN = 1.1348,
30             PISUBK = 0.0027,
31             DELTTF = 0.00906,
32             DFSDT = 0.01,
33             NIGTAB = 19,
34             NAPDVX = 25,
35             NDATA = 3,
36             TPRINT = 0.002,
37             BF1 = 0.1, 0.1, 0.1, 27*1.0,
38             CHC = 1.4, 1.3, 1.2, 1.1, 26*1.0,
39             DELTFF = 0.010, 0.007, 0.005, 27*1.0,
40             DDHC = 0.94,
41             TPSCRI = 850.0,
42             FCRT = 0.1,
43             NBL = 10,
44             NPUNCH = 1,
45             NEPTAB = 20
46             &END
47             2
48             0.0000  0.0000
49             1.0000  1.0000
50             2
51             0.0000  0.0000
52             1.0000  1.0000
53             2
54             0.0000  0.0000
55             1.0000  1.0000
56             2
57             0.0000  0.0000
58             1.0000  1.0000
59             2
60             0.0000  0.0000
61             1.0000  1.0000
62             2
63             0.0000  0.0000
64             1.0000  1.0000
65             2
66             0.0000  0.0000
67             1.0000  1.0000
68             2

```

69	0.0000	0.0000						
70	1.0000	1.0000						
71	2							
72	0.0000	0.0000						
73	1.0000	1.0000						
74	2							
75	0.0000	0.0000						
76	1.0000	1.0000						
77	2							
78	0.0000	0.0000						
79	1.0000	1.0000						
80	2							
81	0.0000	0.0000						
82	1.0000	1.0000						
83	2							
84	0.0000	0.0000						
85	1.0000	1.0000						
86	2							
87	0.0000	0.0000						
88	1.0000	1.0000						
89	2							
90	0.0000	0.0000						
91	1.0000	1.0000						
92	2							
93	0.0000	0.0000						
94	1.0000	1.0000						
95	2							
96	0.0000	0.0000						
97	1.0000	1.0000						
98	2							
99	0.0000	0.0000						
100	1.0000	1.0000						
101	2							
102	0.0000	0.0000						
103	1.0000	1.0000						
104	2							
105	0.0000	0.0000						
106	1.0000	1.0000						
107	0.	1.						
108	0.02	18.11						
109	0.028	46.95						
110	0.044	406.05						
111	0.052	484.35						
112	0.064	514.35						
113	0.07	516.69						
114	0.088	510.06						
115	0.18	427.47						
116	0.24	392.42						
117	0.288	371.00						
118	0.344	348.43						
119	0.356	339.86						
120	0.38	313.75						
121	0.424	275.18						
122	0.46	253.38						
123	0.508	232.73						
124	0.556	219.5						
125	0.566	0.0						
126	2.670	2538.440	1154.450	1154.45	31.	22439.	59741.	2.21
127	57.970	2543.620	1156.490	1156.49				
128	113.280	2561.700	1144.950	1144.95				
129	168.580	2706.870	278.930	278.93				
130	223.890	2612.100	181.180	181.18				
131	279.190	2645.000	182.310	182.31	329.52	877.	20151.	1.5
132	334.500	2658.850	182.790	182.79				
133	389.800	2705.320	184.380	184.38				
134	445.100	2774.530	186.720	186.72				
135	500.410	2844.420	189.060	189.06				
136	555.710	2915.190	191.400	191.40				
137	611.020	3029.790	195.130	195.13	649.52	2896.	20003.	1.5

138	666.320	2757.930	186.160	186.16				
139	721.630	2720.090	184.880	184.88				
140	776.930	2789.300	187.220	187.22				
141	832.230	2859.380	189.560	189.56				
142	887.540	2930.330	191.890	191.89	942.	3653.	19531.	1.5
143	942.840	3072.280	196.490	196.49				
144	998.150	3025.310	194.980	194.98				
145	1053.450	3224.030	201.280	201.28				
146	1108.760	3731.700	216.550	216.55				
147	1164.060	4316.430	232.900	232.9				
148	1219.360	4946.200	249.310	249.31	1293.2	15349.	77964.	9.1
149	1274.670	5404.350	260.600	260.6				
150	1329.972	5594.670	265.150	265.15				
151	0.6808	0.1002						
152	4.7651	0.4383						
153	68.08	1.054						
154	&NAME							
155	TMAX = 0.01,							
156	DELFAC = 1.3							
157	&END							
158	&NAME							
159	TMAX = 0.30							
160	&END							
161	&NAME							
162	TMAX = 0.40,							
163	TPRINT = 0.005							
164	&END							

APPENDIX B
Caveny Program Makefile

```

1      =====
2      # File: makefile
3      =====
4      # This file contains all dependencies necessary to make the cavy code
5      =====
6
7      F77 = f77
8      FFLAGS= -g
9
10     OBJECTS =      \
11         ans.o\
12         brcal.o\
13         fccal.o\
14         fct.o\
15         fct2.o\
16         filt.o\
17         hccal.o\
18         inhomo.o\
19         iterp1.o\
20         lagin2.o\
21         lbc.o\
22         migdot.o\
23         pnch.o\
24         prepc.o\
25         prepp.o\
26         rbc.o\
27         redpun.o\
28         runge.o\
29         sdata.o\
30         sdatac.o\
31         slotk.o\
32         solutc.o\
33         solutp.o\
34         srnbug.o\
35         thrus1.o\
36         timest.o\
37         trid.o\
38         decomp.o\
39         solv.o\
40         gsvolv.o\
41         asolv.o\
42         vsolv.o\
43         repl2.o\
44         repl1.o\
45         sing.o\
46         tscal.o\
47         main.o
48
49     cavy: $(OBJECTS)
50             $(F77) $(FFLAGS) -o cavy $(OBJECTS)

```

APPENDIX C
xplot Source Program


```

69      Push(s);
70      cliNum++;
71      cliIndex=0;
72      XtVaSetValues(cliText,
73          XtNstring,"",XtNinsertPosition,strlen(""),NULL);
74      printf(">cli: %3d %s\n",cliNum,s);
75  }
76  strcpy(inputString,s);
77  BreakdownString(inputString,commandString,&num);
78  DoParse(num,commandString);
79 }
80
81 void RecallCommand(w,event,params,numParams)
82     Widget w; XEvent *event; String *params; Cardinal *numParams;
83 {
84     if(*numParams>0)
85         if(params[0][0]=='1') {
86             if((cliIndex<cliNum)&&(cliIndex<15)) {
87                 cliIndex++;
88                 XtVaSetValues(cliText,XtNstring,cliString[cliIndex],NULL);
89                 XawTextSetInsertionPoint(cliText,strlen(cliString[cliIndex]));
90             }
91         }
92         if(params[0][0]=='2') {
93             if(cliIndex>0) {
94                 cliIndex--;
95                 XtVaSetValues(cliText,XtNstring,cliString[cliIndex],NULL);
96                 XawTextSetInsertionPoint(cliText,strlen(cliString[cliIndex]));
97             }
98         }
99     }
100
101 void Echo(s)
102     String s;
103 {
104     int sLength=strlen(s),code;
105     XawTextBlock text;
106     text.firstPos=0; text.length=sLength; text.ptr=s; text.format=FMT8BIT;
107     code = XawTextReplace(logText,startPos,endPos,&text);
108     startPos = startPos+sLength;
109     endPos = startPos;
110     XawTextSetInsertionPoint(logText,startPos);
111 }
112
113
114 void Push(s)
115     String s;
116 {
117     int i;
118
119     for(i=(NUMcliString-1);i>1;i--)
120         strcpy(cliString[i],cliString[i-1]);
121     strcpy(cliString[1],s);
122     strcpy(cliString[0],"");
123     for (i=0;i<NUMcliString;i++)
124         commandStack[NUMcliString-i-1]=cliString[i];
125     XawListChange(commList,commandStack,15,0,False);
126
127 }
128
129 void BreakdownString(old,new,count)
130     char old[]; char new[]; int *count;
131 {
132     int i=0,m=0,n=0;
133
134     while (old[i]!='\0') {
135         while ((old[i]==cSpace)|| (old[i]==cTab))
136             i++;
137         while ((old[i]!=cSpace)&&(old[i]!=cTab)&&(old[i]!='\0')&&(n<32)) {

```

```

138         new[m][n] = old[i];
139         i++;
140         n++;
141     }
142     new[m][n] = '\0';
143     m++;
144     n = 0;
145 }
146 *count = m;
147 }
148 void DoParse(argc, argv)
149     int argc; char argv[] [32];
150 {
151     int l, m, n;
152
153     if (argc > 0) {
154         if (strcmp(argv[0], "quit") == 0) {
155             exit(0);
156         }
157         else if (strcmp(argv[0], "change") == 0) {
158             printf("(<change> unimplemented)\n");
159         }
160         else if (strcmp(argv[0], "decrement") == 0) {
161             printf("(<dec[rement]> unimplemented)\n");
162         }
163         else if (strcmp(argv[0], "edit") == 0) {
164             printf("(<edit> unimplemented)\n");
165         }
166         else if (strcmp(argv[0], "execute") == 0) {
167             printf("(<exec[ute]> unimplemented)\n");
168         }
169         else if (strcmp(argv[0], "hide") == 0) {
170             if (argc > 2) {
171                 if (strcmp(argv[1], "plot") == 0) {
172                     n = atoi(argv[2]);
173                     if ((n >= 0) && (n <= MAXPLOTS) && (plotExists[n])) {
174                         ShowHide2(dummy, wPlot0 + n, False, 0);
175                     }
176                 }
177             }
178         }
179         else if (strcmp(argv[0], "increment") == 0) {
180             printf("(<inc[rement]> unimplemented)\n");
181         }
182         else if (strcmp(argv[0], "kill") == 0) {
183             if (argc > 1) {
184                 if (strcmp(argv[1], "plot") == 0) {
185                     if (argc > 2) {
186                         n = atoi(argv[2]);
187                         if ((n >= 0) && (n <= MAXPLOTS) && (plotExists[n])) {
188                             KillPlot(n);
189                         }
190                     }
191                 }
192             }
193         }
194         else if (strcmp(argv[0], "load") == 0) {
195             if (argc > 1) {
196                 if (strcmp(argv[1], "data") == 0) {
197                     LoadData(dsName);
198                 }
199             }
200         }
201         else if (strcmp(argv[0], "make") == 0) {
202             if (argc > 2) {
203                 n = atoi(argv[2]);
204                 if ((n >= 0) && (n <= MAXPLOTS) && (!plotExists[n])) {
205                     MakePlot(n);
206                 }
207             }
208         }
209     }
210 }

```

```

207         }
208     }
209     else if (strcmp(argv[0],"print")==0) {
210         printf("{<print> unimplemented}\n");
211     }
212     else if (strcmp(argv[0],"save")==0) {
213         printf("{<save> unimplemented}\n");
214     }
215     else if (strcmp(argv[0],"set")==0) {
216         if (argc>1) {
217             if (strcmp(argv[1],"data")==0) {
218                 if (argc>2) {
219                     strcpy(dsName,argv[2]);
220                     SetStatusLabel();
221                 }
222                 else SelectString(1);
223             }
224             if (strcmp(argv[1],"format")==0) {
225                 if (argc>2) {
226                     strcpy(pfName,argv[2]);
227                     SetStatusLabel();
228                 }
229                 else SelectString(2);
230             }
231         }
232     }
233     else if (strcmp(argv[0],"show")==0) {
234         if (argc>2) {
235             if (strcmp(argv[1],"plot")==0) {
236                 n=atoi(argv[2]);
237                 if ((n>=0)&&(n<=7)&&(plotExists[n]))
238                     ShowHide2(dummy,wPlot0+n,True,0);
239             }
240         }
241     }
242     else if (strcmp(argv[0],"view")==0) {
243         printf("{<view> unimplemented}\n");
244     }
245     else printf("{unrecognized command [%s] [%d]...}\n",argv[0],argc-1);
246 }
247 }
248 */
2
-----.
3     xpData.c
4
5 */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpPlots.h"
11 #include "xpWidgets.h"
12
13 /* $1: Global variables */
14
15 Boolean dataLoaded=False;
16 char columnNames[2*MAXCOLS][24];
17
18 int numPlanes,numCols,numRows,numTimeDep;
19 int currentPlane[MAXPLOTS]={0,0,0,0,0,0,0,0};
20 int    currentRow[MAXPLOTS]={0,0,0,0,0,0,0,0};
21
22 float x[MAXROWS];
23 float z[MAXPLANES];
24
25 float plotData[MAXPLOTS][2*MAXCOLS][MAXPLANES];
26 float dMin[MAXPLOTS][2*MAXCOLS];
27 float dMax[MAXPLOTS][2*MAXCOLS];

```

```

28     float xMin, xMax, xRange;
29     float zMin, zMax, zRange;
30     float yMin[MAXPLOTS], yMax[MAXPLOTS], yRange[MAXPLOTS];
31
32     /* $2: Local variables */
33
34     float d[MAXPLANES][MAXCOLS+1][MAXROWS];
35
36     /* $3: Global functions */
37
38     void LoadData();
39     void PrintData();
40     void CopyData();
41
42     /* $4: Local functions */
43
44     void StatInd();
45     void StatData();
46     void PrintPlotData();
47
48     /* $5: Implementation ===== */
49
50     void LoadData(infile)
51     char *infile;
52     {
53         FILE *fi;
54         int i,j,k,n=0;
55         float dummy;
56         char s[64];
57
58         printf(">Now loading dataset from file: <%s>\n",infile);
59         fi = fopen(infile,"r");
60         if(fi!=NULL) {
61             strcpy(s,"Dataset: ");
62             strcat(s,infile);
63             XtVaSetValues(dataShell,XtNtitle,s,NULL);
64             XtVaSetValues(dataText,XtNstring,infile,NULL);
65             for (i=0;i<8;i++)
66                 XtVaSetValues(PMitem[1][i],XtNsensitive,True,NULL);
67             dataLoaded=True;
68             fscanf(fi,"%d%d%d\n",&numCols,&numRows,&numTimeDep);
69             for (k=0;k<numCols;k++) {
70                 fgets(columnNames[k],25,fi);
71                 for (i=0;i<strlen(columnNames[k]);i++)
72                     if (columnNames[k][i]=='\n') columnNames[k][i]='\0';
73                 XtVaSetValues(FEItem[1][k],XtNlabel,columnNames[k],
74                               XtNsensitive,True,NULL);
75                 XtVaSetValues(CFItem[0][k],XtNlabel,columnNames[k],
76                               XtNsensitive,True,NULL);
77                 XtVaSetValues(DAIItem[0][k],XtNlabel,columnNames[k],NULL);
78             }
79             for (k=0;k<numTimeDep;k++) {
80                 fgets(columnNames[MAXCOLS+k],25,fi);
81                 for (i=0;i<strlen(columnNames[MAXCOLS+k]);i++)
82                     if (columnNames[MAXCOLS+k][i]=='\n')
83                         columnNames[MAXCOLS+k][i]='\0';
84                 XtVaSetValues(FEItem[2][k],XtNlabel,columnNames[MAXCOLS+k],
85                               XtNsensitive,True,NULL);
86                 XtVaSetValues(CFItem[0][MAXCOLS+k],XtNlabel,columnNames[MAXCOLS+k],
87                               XtNsensitive,True,NULL);
88                 XtVaSetValues(DAIItem[0][8+k],XtNlabel,columnNames[MAXCOLS+k],NULL);
89             }
90             if ((numRows>0) && (numCols>0)) {
91                 k=0;
92                 do {
93                     n=fscanf(fi,"%e",&z[k]);
94                     if (n!=EOF) {
95                         for(i=0;i<numRows;i++) {
96                             if(k==0)

```

```

97             fscanf(fi,"%e",&x[i]);
98         else
99             fscanf(fi,"%e",&dummy);
100        for(j=0;j<numCols;j++)
101            fscanf(fi,"%e",&d[k][j][i]);
102        }
103        for (i=0;i<numTimeDep;i++)
104            fscanf(fi,"%e",&d[k][MAXCOLS][i]);
105        }
106        k++;
107    } while ( (n!=EOF)&&(k<(MAXPLANES-1)) );
108
109    numPlanes=k-1;
110    fclose(fi);
111    StatInd();
112    } else printf("Unable to load dataset (does not exist)\n");
113
114}
115
116 void PrintData(outfile)
117     char *outfile;
118 {
119     FILE *fo;
120     int i,j,k;
121
122     printf(">Now printing dataset to file: <%s>\n",outfile);
123     fo = fopen(outfile,"w");
124     fprintf(fo,"%4d %4d %4d\n",numPlanes,numCols,numRows);
125     if ((numRows>0) && (numCols>0) && (numPlanes>0))
126     for(k=0;k<numPlanes;k++) {
127         fprintf(fo,"%12.6f\n",z[k]);
128         for(i=0;i<numRows;i++) {
129             fprintf(fo,"%12.6f",x[i]);
130             for(j=0;j<numCols;j++)
131                 fprintf(fo,"%12.6f",d[k][j][i]);
132             fprintf(fo,"\n");
133         }
134     }
135     fclose(fo);
136 }
137
138 void CopyData(plotNum)
139     int plotNum;
140 {
141     int i,j;
142     if (planar[plotNum])
143         for(j=0;j<numCols;j++)
144             for(i=0;i<numRows;i++)
145                 plotData[plotNum][j][i]=d[currentPlane[plotNum]][j][i];
146     else {
147         for(j=0;j<numCols;j++)
148             for(i=0;i<numPlanes;i++)
149                 plotData[plotNum][j][i]=d[i][j][currentRow[plotNum]];
150             for(j=0;j<numTimeDep;j++)
151                 for(i=0;i<numPlanes;i++)
152                     plotData[plotNum][MAXCOLS+j][i]=d[i][MAXCOLS][j];
153     }
154 }
155
156
157 void StatData(p)
158     int p;
159 {
160     int i,j,m,n,test;
161     float Max,Min;
162     int count;
163
164     count = (planar[p] ? numRows : numPlanes);
165     test = (planar[p] ? MAXCOLS : 2*MAXCOLS);

```

```

166     for(j=0;j<test;j++) {
167         if (useCol[p][j]) {
168             Min= 1000000;
169             Max=-1000000;
170             for(i=0;i<count;i++) {
171                 if(plotData[p][j][i]<Min) Min=plotData[p][j][i];
172                 if(plotData[p][j][i]>Max) Max=plotData[p][j][i];
173             }
174             dMin[p][j]=Min;
175             dMax[p][j]=Max;
176         }
177     }
178     Min= 1000000;
179     Max=-1000000;
180     for(j=0;j<test;j++) {
181         if (useCol[p][j]) {
182             if (dMin[p][j]<Min) Min=dMin[p][j];
183             if (dMax[p][j]>Max) Max=dMax[p][j];
184         }
185         yMin[p]=Min;
186         yMax[p]=Max;
187         yRange[p]=yMax[p]-yMin[p];
188         if (yRange[p]<=0.001) {
189             yRange[p]=1.0;
190             yMax[p]=yMin[p]+1.0;
191         }
192     }
193 }
194 void StatInd()
195 {
196     int i;
197     xMin= 1000000;
198     xMax=-1000000;
199     for(i=0;i<numRows;i++) {
200         if(x[i]<xMin) xMin=x[i];
201         if(x[i]>xMax) xMax=x[i];
202     }
203     xRange=xMax-xMin;
204     zMin= 1000000;
205     zMax=-1000000;
206     for(i=0;i<numPlanes;i++) {
207         if(z[i]<zMin) zMin=z[i];
208         if(z[i]>zMax) zMax=z[i];
209     }
210     zRange=zMax-zMin;
211 }
212
213 void PrintPlotData(p)
214 {
215     int p;
216     {
217         int i,j,k;
218         printf(">Now printing plotset(%d)...\\n",p);
219         printf("%4d %4d %4d\\n",currentPlane[p],numCols,currentRow[p]);
220         if (planar[p]) {
221             if ((numRows>0) && (numCols>0))
222                 printf("%10.4f\\n",z[currentPlane[p]]);
223                 for(i=0;i<numRows;i++) {
224                     printf("%10.4f",x[i]);
225                     for(j=0;j<numCols;j++)
226                         printf("%10.4f",plotData[p][j][i]);
227                     printf("\\n");
228                 }
229                 printf("xMin: %10.4f\\n",xMin);
230                 printf("xMax: %10.4f\\n",xMax);
231                 printf("xRange: %10.4f\\n",xRange);
232             }
233         else {
234

```

```

235     if ((numPlanes>0) && (numCols>0))
236         printf("%10.4f\n",x[currentRow[p]]);
237     for(i=0;i<numPlanes;i++) {
238         printf("%10.4f",z[i]);
239         for(j=0;j<numCols;j++)
240             printf("%10.4f",plotData[p][j][i]);
241         printf("\n");
242     }
243     printf("zMin: %10.4f\n",zMin);
244     printf("zMax: %10.4f\n",zMax);
245     printf("zRange: %10.4f\n",zRange);
246 }
247 printf("yMin: %10.4f\n",yMin[p]);
248 printf("yMax: %10.4f\n",yMax[p]);
249 printf("yRange: %10.4f\n",yRange[p]);
250 }
251 */
2 -----
3     xpFormats.c
4 -----
5 */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpgeneral.h"
11 #include "xpwidgets.h"
12 #include "xpmenus.h"
13 #include "xpdata.h"
14 #include "xplots.h"
15
16 #define feLoad 1
17 #define feSave 2
18 #define feCancel 3
19 #define feAccept 4
20 #define feRevert 5
21
22 /* $1: Global variables */
23
24 PlotFormat f[MAXPLOTS];
25 int activeFormat=0;
26
27 /* $2: Local variables - (none) */
28
29 /* $3: Global functions */
30
31 void InitFormat();
32 void SetFEValues();
33 void GetFEValues();
34 void RecalcFormat();
35 void FEToggleCallback();
36 void FECCommandCallback();
37 void SetFEValues();
38 void GetFEValues();
39
40 /* $4: Local functions - (none) */
41
42 /*
43 void PrintFormat();
44 */
45
46 /* $5: Implementation ===== */
47
48 void InitFormat(p)
49     int p;
50 {
51     int i,j;
52

```

```

53     sprintf(f[p].title,"Plot #%d",p);
54     for (i=0;i<8;i++) {
55         f[p].color[i]=i;
56         f[p].color[MAXCOLS+i]=MAXCOLS+i;
57         f[p].numDashes[i]=0;
58         for (j=0;j<8;j++)
59             f[p].dashes[i][j]=0;
60         f[p].opt[i]=False;
61     }
62     f[p].borderL=64; f[p].borderR=32; f[p].borderT=32; f[p].borderB=64;
63     f[p].xNum=10; f[p].yNum=10;
64     f[p].xMin=f[p].xMax=0; f[p].yMin=f[p].yMax=0;
65     strcpy(f[p].xFormat,".0*"); strcpy(f[p].yFormat,".0*");
66     f[p].opt[aScaleAxes]=True; f[p].opt[aSmoothNum]=False;
67     f[p].xTick=4; f[p].yTick=4;
68     f[p].width=640; f[p].height=480;
69     f[p].xInc=(f[p].width -f[p].borderL-f[p].borderR)/f[p].xNum;
70     f[p].yInc=(f[p].height-f[p].borderT-f[p].borderB)/f[p].yNum;
71     f[p].nGrid=f[p].xNum+f[p].yNum+2;
72     f[p].xLength=f[p].xNum*f[p].xInc;
73     f[p].yLength=f[p].yNum*f[p].yInc;
74 }
75
76 void SetFEValues(p)
77     int p;
78 {
79     char s[16][32],t[16];
80     int i,j;
81
82     strcpy(s[0],f[p].title);
83     XtVaSetValues(FEItem[FEBTitle][0],XtNstring,s[0],NULL);
84
85     for (i=0;i<(2*MAXCOLS);i++) {
86         sprintf(s[i],"%d",f[p].color[i]);
87         XtVaSetValues(DAIItem[1][i],XtNstring,s[i],NULL);
88         XtVaSetValues(DAIItem[0][i],
89             XtNbackground,lineColor[f[p].color[i]],
90             XtNforeground,rColor[Black].pixel,NULL);
91     }
92
93     for (i=0;i<(2*MAXCOLS);i++) {
94         strcpy(s[i],"");
95         for (j=0;j<f[p].numDashes[i];j++) {
96             sprintf(t,"%d ",(int) f[p].dashes[i][j]);
97             strcat(s[i],t);
98         }
99         XtVaSetValues(DAIItem[2][i],XtNstring,s[i],NULL);
100    }
101
102     sprintf(t,"%%%sf",f[p].xFormat);
103     sprintf(s[0],t,f[p].xMin);
104     sprintf(s[1],t,f[p].xMax);
105
106     sprintf(s[2],"%d",f[p].xNum);
107     sprintf(s[3],"%d",f[p].xTick);
108     strcpy(s[4],f[p].xFormat);
109     sprintf(s[5],"%d",f[p].borderL);
110     sprintf(s[6],"%d",f[p].borderR);
111     for (i=0;i<7;i++)
112         XtVaSetValues(FEItem[FEBX][i],XtNstring,s[i],NULL);
113
114     sprintf(t,"%%%sf",f[p].yFormat);
115     sprintf(s[0],t,f[p].yMin);
116     sprintf(s[1],t,f[p].yMax);
117     sprintf(s[2],"%d",f[p].yNum);
118     sprintf(s[3],"%d",f[p].yTick);
119     strcpy(s[4],f[p].yFormat);
120     sprintf(s[5],"%d",f[p].borderT);
121     sprintf(s[6],"%d",f[p].borderB);

```

```

122     for (i=0;i<7;i++)
123         XtVaSetValues(FEItem[FEBY][i],XtNstring,s[i],NULL);
124
125     for(i=0;i<MAXCOLS;i++)
126         if (i<numCols)
127             XtVaSetValues(FEItem[FEBVars][i],XtNsensitive,True,NULL);
128         else
129             XtVaSetValues(FEItem[FEBVars][i],XtNsensitive,False,NULL);
130     for(i=0;i<MAXCOLS;i++)
131         if (i<numTimeDep)
132             XtVaSetValues(FEItem[2][i],XtNsensitive,True,NULL);
133         else
134             XtVaSetValues(FEItem[2][i],XtNsensitive,False,NULL);
135     for(i=0;i<numCols;i++)
136         XtVaSetValues(FEItem[FEBVars][i],XtNstate,useCol[p][i],NULL);
137     for(i=0;i<numTimeDep;i++)
138         XtVaSetValues(FEItem[2][i],XtNstate,useCol[p][MAXCOLS+i],NULL);
139     for(i=0;i<8;i++)
140         XtVaSetValues(FEItem[FEBOpts][i],XtNstate,f[p].opt[i],NULL);
141 }
142
143 void GetFEValues(p)
144     int p;
145 {
146     String t;
147     char s[16][16];
148     int i,j,count,u[8];
149
150     XtVaGetValues(FEItem[0][0],XtNstring,&t,NULL);
151     strcpy(f[p].title,t);
152
153     for (i=0;i<(2*MAXCOLS);i++) {
154         XtVaGetValues(DAItem[1][i],XtNstring,&t,NULL);
155         strcpy(s[i],t);
156         j = atoi(s[i]);
157         f[p].color[i] = ( (j>=0)&&(j<=31) ? j : i);
158     }
159
160     for (i=0;i<(2*MAXCOLS);i++) {
161         XtVaGetValues(DAItem[2][i],XtNstring,&t,NULL);
162         strcpy(s[i],t);
163         count=sscanf(s[i],"%d %d %d %d %d %d %d",
164             &u[0],&u[1],&u[2],&u[3],&u[0],&u[1],&u[2],&u[3]);
165         f[p].numDashes[i]=count;
166         for (j=0;j<8;j++)
167             f[p].dashes[i][j]=(char) u[j];
168     }
169
170     for (i=0;i<7;i++) {
171         XtVaGetValues(FEItem[FEBX][i],XtNstring,&t,NULL);
172         strcpy(s[i],t);
173     }
174     f[p].xMin=atoi(s[0]);
175     f[p].xMax=atoi(s[1]);
176     f[p].xNum=atoi(s[2]);
177     f[p].xTick=atoi(s[3]);
178     strncpy(f[p].xFormat,s[4],15);
179     f[p].borderL=atoi(s[5]);
180     f[p].borderR=atoi(s[6]);
181
182     for (i=0;i<7;i++) {
183         XtVaGetValues(FEItem[FEBY][i],XtNstring,&t,NULL);
184         strcpy(s[i],t);
185     }
186     f[p].yMin=atoi(s[0]);
187     f[p].yMax=atoi(s[1]);
188     f[p].yNum=atoi(s[2]);
189     f[p].yTick=atoi(s[3]);
190     strncpy(f[p].yFormat,s[4],15);

```

```

191     f[p].borderT=atoi(s[5]);
192     f[p].borderB=atoi(s[6]);
193
194     for(i=0;i<numCols;i++)
195         XtVaGetValues(FEItem[FEBVars][i],XtNstate,&useCol[p][i],NULL);
196     for(i=0;i<numTimeDep;i++)
197         XtVaGetValues(FEItem[2][i],XtNstate,&useCol[p][MAXCOLS+i],NULL);
198     for(i=0;i<8;i++)
199         XtVaGetValues(FEItem[FEBOpts][i],XtNstate,&f[p].opt[i],NULL);
200
201 }
202
203 void RecalcFormat(p)
204     int p;
205 {
206     f[p].xInc=(f[p].width -f[p].borderL-f[p].borderR)/f[p].xNum;
207     f[p].yInc=(f[p].height-f[p].borderT-f[p].borderB)/f[p].yNum;
208     f[p].nGrid=f[p].xNum+f[p].yNum+2;
209     f[p].xLength=f[p].xNum*f[p].xInc;
210     f[p].yLength=f[p].yNum*f[p].yInc;
211 /*PrintFormat(p); */
212 }
213
214 /*
215 void PrintFormat(p)
216     int p;
217 {
218     printf("\n");
219     printf("    width:    %d\n",f[p].width);
220     printf("    height:   %d\n",f[p].height);
221     printf("    xInc:     %d\n",f[p].xInc);
222     printf("    yInc:     %d\n",f[p].yInc);
223     printf("    xLength:  %d\n",f[p].xLength);
224     printf("    yLength:  %d\n",f[p].yLength);
225 }
226 */
227
228 void LoadFormat()
229 {
230     FILE *fi;
231     char s[64],*t;
232     int i,j,a=activeFormat;
233
234     XtVaGetValues(FEItem[9][0],XtNstring,&t,NULL);
235     strcpy(s,t);
236
237     fi = fopen(s,"r");
238     if (fi!=NULL) {
239         fgets(t,63,fi);
240         strcpy(s,t);
241         if(strlen(s)>1) strncpy(f[a].title,s,strlen(s)-1);
242         fgets(t,63,fi);
243         strcpy(s,t);
244         if(strlen(s)>1) strncpy(f[a].xFormat,s,strlen(s)-1);
245         fgets(s,63,fi);
246         strcpy(s,t);
247         if(strlen(s)>1) strncpy(f[a].yFormat,s,strlen(s)-1);
248
249         for (i=0;i<8;i++) {
250             fscanf(fi,"%d",&f[a].color[i]);
251             printf("%d ",f[a].color[i]);
252         }
253         printf("\n");
254         for (i=0;i<8;i++) {
255             fscanf(fi,"%d",&j);
256             f[a].opt[i] = ( (j==0) ? False : True );
257         }
258         fscanf(fi,"%d %d",&f[a].xNum,&f[a].yNum);
259         fscanf(fi,"%e %e %e %e",&f[a].xMin,&f[a].xMax,&f[a].yMin,&f[a].yMax);

```

```

260     fscanf(fi,"%d %d",&f[a].xTick,&f[a].yTick);
261     fscanf(fi,"%d %d %d %d",
262             &f[a].borderL,&f[a].borderR,&f[a].borderT,&f[a].borderB);
263     fclose(fi);
264     SetFEValues(a);
265 }
266
267 }
268
269 void SaveFormat()
270 {
271     FILE *fo;
272     char s[64],*t;
273     int i,a=activeFormat;
274     XtVaGetValues(FEItem[9][0],XtNstring,&t,NULL);
275     strcpy(s,t);
276
277     if (s[0]!=NULL) {
278         fo = fopen(s,"w");
279         fprintf(fo,"%s\n",f[a].title);
280         fprintf(fo,"%s\n%s\n",f[a].xFormat,f[a].yFormat);
281         for (i=0;i<8;i++)
282             fprintf(fo,"%d ",f[a].color[i]);
283         fprintf(fo,"\n");
284
285         for (i=0;i<8;i++)
286             fprintf(fo,"%d ",f[a].opt[i]);
287         fprintf(fo,"\n");
288
289         fprintf(fo,"%d %d\n",f[a].xNum,f[a].yNum);
290         fprintf(fo,"%f %f %f\n",f[a].xMin,f[a].xMax,f[a].yMin,f[a].yMax);
291         fprintf(fo,"%d %d\n",f[a].xTick,f[a].yTick);
292         fprintf(fo,"%d %d %d %d\n",
293                 f[a].borderL,f[a].borderR,f[a].borderT,f[a].borderB);
294
295         fclose(fo);
296     }
297 }
298
299 void FEPAGECallback(w,client_data,call_data)
300     Widget w; XtPointer client_data,call_data;
301 {
302     int i = (int) client_data;
303     char s[64];
304
305     switch (i) {
306         case 0:
307             if (activeFormat<(MAXPLOTS-1)) {
308                 activeFormat++;
309                 SetFEValues(activeFormat);
310                 sprintf(s,"Format #d",activeFormat);
311                 XtVaSetValues(FETop[FEBPage],XtNlabel,s,NULL);
312             }
313             break;
314         case 1:
315             if (activeFormat>0) {
316                 activeFormat--;
317                 SetFEValues(activeFormat);
318                 sprintf(s,"Format #d",activeFormat);
319                 XtVaSetValues(FETop[FEBPage],XtNlabel,s,NULL);
320             }
321             break;
322     }
323 }
324
325 void FECommandCallback(w,client_data,call_data)
326     Widget w; XtPointer client_data,call_data;
327 {

```

```

329     int i = (int) client_data;
330
331     switch (i) {
332         case feLoad:
333             LoadFormat();
334             break;
335         case feSave:
336             SaveFormat();
337             break;
338         case feCancel:
339             ShowHide2(dummy,wFormat,True,0);
340             SetFEValues(activeFormat);
341             break;
342         case feAccept:
343             GetFEValues(activeFormat);
344             if (plotExists[activeFormat]) {
345                 RecalcFormat(activeFormat);
346                 StatData(activeFormat);
347                 SetupGrid(activeFormat);
348                 SetupAxisLabels(activeFormat);
349                 SetupPoints(activeFormat);
350                 RedrawPlot2(activeFormat);
351             }
352             break;
353         case feRevert:
354             SetFEValues(activePlot);
355             break;
356     }
357 }
358 */
1 -----
2 ----- xpGeneral.c
3 -----
4 -----
5 */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpwidgets.h"
11 #include "xpcommands.h"
12 #include "xpdata.h"
13 #include "xpplots.h"
14
15 /* $1: Global variables */
16
17 Screen *screen;
18 Display *display;
19 Boolean windowUp[48];
20
21 GC gc[MAXGCS];
22 XColor exact,color[16],rColor[16];
23 Font f6x10,f6x12,f6x13,fHelv18b;
24 XFontStruct *fs6x10,*fs6x12,*fs6x13,*fsHelv18b;
25
26 XawTextPosition startPos=0,endPos=0;
27 char inName[64]={" "},dsName[64]={" "},pfName[64]={" "},plName[4]={" "};
28
29 /* $2: Local variables - (none) */
30
31 /* $3: Global functions */
32
33 void LoadFonts();
34 void LoadColors();
35 void InitData();
36 void ProcessKey();
37 void Idle();
38
39 /* $4: Local functions - (none) */

```

```

40  /* $5: Implementation ===== */
41
42 void LoadFonts()
43 {
44     f6x10     = XLoadFont(display, "6x10");
45     f6x12     = XLoadFont(display, "6x12");
46     f6x13     = XLoadFont(display, "6x13");
47     fHelv18b  = XLoadFont(display, "-*-helvetica-bold-r-*--18-180-*-*");
48
49     fs6x10   = XQueryFont(display, f6x10);
50     fs6x12   = XQueryFont(display, f6x12);
51     fs6x13   = XQueryFont(display, f6x13);
52     fsHelv18b = XQueryFont(display, fHelv18b);
53 }
54
55 void LoadColors()
56 {
57     Colormap cmap;
58     Status r;
59     FILE *fi;
60     char s[64], t[64], *u, *v;
61     int i, count=0;
62
63     cmap = XDefaultColormap(display, DefaultScreen(display));
64
65
66     fi = fopen("XPlot.clr", "r");
67     if (fi!=NULL) {
68         while (fscanf(fi, "%d %s", &i, s)!=EOF) {
69             printf("-- %d %s\n", i, s);
70             if ((i>0)&&(i<=31)) {
71                 r=XAllocNamedColor(display, cmap, s, &exact, &color[i]);
72             }
73         }
74         fclose(fi);
75     }
76
77     r=XAllocNamedColor(display, cmap, "Black" ,&exact, &rColor[Black]);
78     r=XAllocNamedColor(display, cmap, "White" ,&exact, &rColor[White]);
79     r=XAllocNamedColor(display, cmap, "Red"   ,&exact, &rColor[Red]);
80     r=XAllocNamedColor(display, cmap, "Green" ,&exact, &rColor[Green]);
81     r=XAllocNamedColor(display, cmap, "Blue"  ,&exact, &rColor[Blue]);
82     r=XAllocNamedColor(display, cmap, "Yellow", &exact, &rColor[Yellow]);
83
84     for (i=0; i<(2*MAXCOLS); i++)
85         lineColor[i]=color[i].pixel;
86
87 }
88
89 void InitData()
90 {
91     int i, j, k;
92
93     for(k=0; k<MAXPLOTS; k++)
94         for(j=0; j<(2*MAXCOLS); j++)
95             useCol[k][j]=False;
96
97     for(i=0; i<NUMcliString; i++)
98         strcpy(cliString[i], "");
99
100    for(i=0; i<MAXPLOTS; i++)
101        plotExists[i]=False;
102
103 }
104
105 void InitGCs()
106 {
107     Window wind = XtWindow(topLevel);
108

```

```

109     int i;
110
111     for(i=0;i<4;i++)
112         gc[i]=XCreateGC(display,wind,0,0);
113
114     XSetForeground(display,gc[0],rColor[Black].pixel);
115     XSetFont(display,gc[0],f6x10);
116
117     XSetForeground(display,gc[1],rColor[Black].pixel);
118     XSetFont(display,gc[1],f6x12);
119
120     XSetForeground(display,gc[2],rColor[Black].pixel);
121     XSetFont(display,gc[2],fHelv18b);
122
123 }
124
125 void DoKey(w, event, params, numParams)
126     Widget w; XEvent *event; String *params; Cardinal *numParams;
127 {
128     XKeyEvent *e = (XKeyEvent*) event;
129 }
130
131 void Idle(w, event, params, numParams)
132     Widget w; XEvent *event; String *params; Cardinal *numParams;
133 {
134 }
135
1 /*
2 -----
3     xpGeometry.c
4 -----
5 */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpWidgets.h"
11 #include "xpGeneral.h"
12 #include "xpCommands.h"
13 #include "xpInput.h"
14
15 #define pi 3.1415926535
16
17 /* $1: Global variables - (none) */
18
19 /* $2: Local variables */
20
21 int activeGeometry=-1;
22 float geomVars[8];
23
24 static String MGInputLabels[6][8] = {
25     { "Web Thickness", "Internal Radius", "Half-Angle", "Epsilon",
26       "Fillet Radius", " ", " ", " " }
27 };
28
29 static String MGOutputLabels[6][8] = {
30     { "Initial Area", "Final Area", " ", " ", " ", " ", " ", " " }
31 };
32
33 /* $3: Global functions */
34
35 void MGCallback();
36 void ChangeMGLabels();
37 void ResizeGeometry();
38 void ExposeGeometry();
39
40 /* $4: Local functions */
41 void RecalcGeometry();
42 void RedrawGeometry();

```

```

43  /* $5: Implementation ===== */
44
45
46  void MGCallback(w,client_data,call_data)
47  {
48      Widget w; XtPointer client_data,call_data;
49
50      int n = (int) client_data;
51      int m;
52      char s[64],*t;
53
54      switch (n) {
55          case 0:
56              m = (int) XawToggleGetCurrent(MGItem[2][0]);
57              if (m!=NULL) {
58                  activeGeometry=m-1;
59                  RecalcGeometry();
60              }
61              break;
62          case 1:
63              ShowHide2(dummy,wMotorConfiguration,True,0);
64              break;
65          case 2:
66              ShowHide2(dummy,wMotorGeometry,False,0);
67              break;
68      }
69
70  }
71
72  void ChangeMGLabels(n)
73  {
74      int i;
75
76      n = 0;
77      for (i=0;i<8;i++)
78          XtVaSetValues(MGItem[0][i],XtNlabel,MGInputLabels[n][i],NULL);
79      for (i=0;i<2;i++)
80          XtVaSetValues(MGItem[3][i],XtNlabel,MGOutputLabels[n][i],NULL);
81
82  }
83
84
85  void RecalcGeometry()
86  {
87      int i,j,k;
88      char s[64],*t;
89      float re,ri,rr,w,bp;
90
91
92      for (i=0;i<8;i++) {
93          XtVaGetValues(MGItem[1][i],XtNstring,&t,NULL);
94          strcpy(s,t);
95          sscanf(s,"%e",&geomVars[i]);
96      }
97
98      switch (activeGeometry) {
99          case 0:
100              re=geomVars[0]; ri=geomVars[1]; w=re-ri;
101              bp=2*ri*pi;
102              break;
103          case 1:
104              re=geomVars[0]; ri=geomVars[1]; rr=geomVars[2]; w=re-ri;
105              bp=2*ri*pi+2*rr*pi;
106              break;
107          case 2:
108              break;
109      }
110      sprintf(s,"%4f",bp);
111

```

```

112     XtVaSetValues(MGItem[4][0], XtNstring, s, NULL);
113 }
115
116 void ExposeGeometry(w, event, params, numParams)
117   Widget w; XEvent *event; String *params; Cardinal *numParams;
118 {
119   XExposeEvent *myevent = (XExposeEvent*) event;
120   static int i=0;
122
123   if ((*myevent).count==0) {
124     i++;
125     printf(":ExposeGeometry (%d)...\\n",i);
126     RedrawGeometry();
127   }
128 }
129
130 void ResizeGeometry(widget, event, params, numParams)
131   Widget widget; XEvent *event; String *params; Cardinal *numParams;
132 {
133   static Dimension oldWidth=360,oldHeight=360;
134   Dimension w,h;
135
136   XtVaGetValues(MGPict, XtNwidth, &w, XtNheight, &h, NULL);
137   if ((w<=oldWidth)&&(h<=oldHeight)) {
138     printf(":ResizeGeometry calling RedrawGeometry...\\n");
139     RedrawGeometry();
140   }
141
142   oldWidth=w;
143   oldHeight=h;
144 }
145
146 void RedrawGeometry()
147 {
148   Dimension w,h;
149   Window wind = XtWindow(MGPict);
150
151   printf(":RedrawGeometry...\\n");
152
153   XtVaGetValues(MGPict, XtNwidth, &w, XtNheight, &h, NULL);
154   XClearWindow(display,wind);
155
156   XSetLineAttributes(display,gc[0],4,LineSolid,CapButt,JoinRound);
157   XDrawArc(display,wind,gc[0],4,4,w-8,h-8,0,360*64);
158   XSetLineAttributes(display,gc[0],0,LineSolid,CapButt,JoinRound);
159
160   switch (activeGeometry) {
161     case 0:
162       break;
163   }
164
165 }
166
167 /*
2-----*
3-----* xpInput.c
4-----*
5-----*/
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpWidgets.h"
11 #include "xpGeneral.h"
12 #include "xpMenus.h"
13
14 #define MAXVARS 44

```

```

15  #define NDELX 0
16  #define TPI 1
17  #define PAM 2
18  #define NINERT 3
19  #define UNIT 4
20  #define AT 5
21  #define XP 6
22  #define XE 7
23  #define XG 8
24  #define GAMA 9
25  #define W 10
26  #define TIGN 11
27  #define TFREF 12
28  #define RUFSUR 13
29  #define DDRG 14
30  #define DDHC 15
31  #define FKPR 16
32  #define ROPR 17
33  #define CPR 18
34  #define TOREF 19
35  #define SIGP 20
36  #define TPSCRI 21
37  #define RREF 22
38  #define PREF 23
39  #define BREXP 24
40  #define EBC 25
41  #define EBEX 26
42  #define DE 27
43  #define CM 28
44  #define ALFAD 29
45  #define EROAT 30
46  #define EROEXP 31
47  #define TMAX 32
48  #define DELTAT 33
49  #define TPRINT 34
50  #define PZONE 35
51  #define LAMBDA 36
52  #define NIGTAB 37
53  #define NAPDVX 38
54  #define NDATA 39
55  #define NPPR 40
56  #define ITABLE 41
57  #define GTABLE 42
58  #define BTABLE 43
59
60  /* $1: Global Variables */
61  char varName[MAXVARS][32], varDesc[MAXVARS][80];
62  char varNumb[MAXVARS][12], varInfo[MAXVARS][800];
63  float varValue[MAXVARS];
64  float iTable[30][2], gTable[30][8], bTable[30][2];
65
66  /* $2: Local Variables */
67
68  int numVars, inputIndex=0;
69  char *IEStrings[100], IEEntries[100][20];
70
71  /* $3: Global Functions */
72
73  void LoadInputData();
74  void IECallback();
75  void IEListCallback();
76  void WriteInputFile();
77  void AcceptInputEntry();
78
79  /* $4: Local Functions */
80
81  void GetInputValues();
82  void SetInputValues();
83

```

```

84     void IncDecInput();
85
86     /* $5: Implementation ===== */
87
88     void LoadInputData()
89     {
90         FILE* fi;
91         int i,j=0;
92         char s[64];
93
94         fi=fopen("XPlot.var","r");
95
96         do {
97             i=fscanf(fi,"%*[ \t\n]");
98             if (i!=EOF) {
99                 i=fscanf(fi,"%*[^\n]/\n",varName[j]);
100                i=fscanf(fi,"%*[^\n]/\n",varDesc[j]);
101                i=fscanf(fi,"%*[^\n]/\n",varNumb[j]);
102                i=fscanf(fi,"%[^"]\"",varInfo[j]);
103            }
104            j++;
105        } while ( (i!=EOF)&&(i!=0)&&(j<MAXVARS) );
106        numVars=j-1;
107
108        for (j=0;j<numVars;j++)
109            i=sscanf(varNumb[j],"%e",&varValue[j]);
110        printf("\n");
111
112        for (i=0;i<=BTABLE;i++) {
113            if (i<ITABLE)
114                sprintf(IEEntries[i],"%-6s = %10.4f",varName[i],varValue[i]);
115            else
116                sprintf(IEEntries[i],"----- %c%s -----",varName[i][0],"TABLE");
117            IEStrings[i] = IEEntries[i];
118            printf("%2d %s\n",i,IEEntries[i]);
119        }
120        printf("\n");
121
122        XawListChange(IEList,IEStrings,BTABLE+1,0,False);
123    }
124
125
126
127    void DoInput()
128    {
129        int i,j,k;
130        char s[64],*t,*u,*v;
131
132        ShowHide2(dummy,1000,True,0);
133    }
134
135    void AcceptInputEntry(w,event,params,numParams)
136        Widget w; XEvent *event; String *params; Cardinal *numParams;
137    {
138        if(numParams>0) {
139            if(params[0][0]=='+')
140                IncDecInput(True);
141            if(params[0][0]=='-')
142                IncDecInput(False);
143        }
144    }
145
146    void GetInputValues(i)
147    {
148        int i;
149        int j,k;
150        char s[2500],*t;
151
152        if ( (i>=0)&&(i<ITABLE) ) {

```

```

153     XtVaGetValues(IEItem[3][0], XtNstring, &t, NULL);
154     strcpy(s, t);
155     sscanf(t, "%e", &varValue[i]);
156     sprintf(IEEntries[i], "%-6s = %10.4f", varName[i], varValue[i]);
157     XawListUnhighlight(IEList);
158 }
159 }
160
161 void SetInputValues(i)
162 {
163     int j, k;
164     char s[2500], t[80];
165
166     if ( (i>=0)&&(i<ITABLE) ) {
167         XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
168         XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
169         XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
170         sprintf(s, "%-2f", varValue[i]);
171         XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
172         XawListHighlight(IEList, i);
173     }
174
175     if ( i==ITABLE ) {
176         XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
177         XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
178         XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
179         strcpy(s, "");
180         k=varValue[NIGTAB];
181         for (j=0;j<k;j++) {
182             sprintf(t, "%10.4f%10.4f\n", iTable[j][0], iTable[j][1]);
183             strncat(s, t);
184         }
185         XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
186         XawListHighlight(IEList, i);
187     }
188
189     if ( i==GTABLE ) {
190         XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
191         XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
192         XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
193         strcpy(s, "");
194         k=varValue[NAPDVX];
195         strcpy(s, "");
196         strcpy(s, "");
197         for (j=0;j<k;j++) {
198             sprintf(t, "%10.4f%10.4f%10.4f%10.4f",
199                     gTable[j][0], gTable[j][1], gTable[j][2], gTable[j][3]);
200             strncat(s, t);
201             if (gTable[j][4]>0) {
202                 sprintf(t, "%10.4f%10.4f%10.4f%10.4f",
203                         gTable[j][4], gTable[j][5], gTable[j][6], gTable[j][7]);
204                 strncat(s, t);
205             }
206             strncat(s, "\n");
207         }
208         XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
209
210         XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
211
212         XawListHighlight(IEList, i);
213
214     }
215
216     if ( i==BTABLE ) {
217         XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
218         XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
219         XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
220         strcpy(s, "");
221         k=varValue[NIGTAB];

```

```

222         for (j=0;j<k;j++) {
223             sprintf(t,"%10.4f%10.4f\n",bTable[j][0],bTable[j][1]);
224             strcat(s,t);
225         }
226         XtVaSetValues(IEItem[3][0],XtNstring,s,NULL);
227         XtVaSetValues(IEItem[3][0],XtNstring,s,NULL);
228         XawListHighlight(IEList,i);
229     }
230 }
231
232 void IECallback(w,client_data,call_data)
233     Widget w; XtPointer client_data,call_data;
234 {
235     int n = (int) client_data;
236     char s[128],*t;
237
238     switch (n) {
239         case 0: /* Start | Restart */
240             inputIndex=0;
241             SetInputValues(inputIndex);
242             break;
243         case 1: /* Insert */
244             break;
245         case 2: /* Prev */
246             IncDecInput(False);
247             break;
248         case 3: /* Next */
249             IncDecInput(True);
250             break;
251         case 4: /* Write */
252             XtVaGetValues(IEItem[4][0],XtNstring,&t,NULL);
253             strcpy(s,t);
254             if (strcmp(s,"")!=0) WriteInputFile(s);
255             break;
256         case 5: /* Cancel */
257             ShowHide2(dummy,wInput,False,0);
258             inputIndex=0;
259             SetInputValues(inputIndex);
260             break;
261     }
262 }
263
264 void IEListCallback(w,client_data,call_data)
265     Widget w; XtPointer client_data,call_data;
266 {
267     XawListReturnStruct *r = (XawListReturnStruct*) call_data;
268     char s[64];
269     int n;
270
271     n = (*r).list_index;
272     printf("Element [%02d] selected from IEList\n",n);
273     GetInputValues(inputIndex);
274     inputIndex=n;
275     SetInputValues(inputIndex);
276 }
277
278 void IncDecInput(inc)
279     Boolean inc;
280 {
281     char s[1024],*t;
282
283     if ( (inc)&&(inputIndex<(BTABLE)) ) {
284         GetInputValues(inputIndex);
285         inputIndex++;
286         SetInputValues(inputIndex);
287     }
288
289     if ( (!inc)&&(inputIndex>0) ) {

```

```

291     GetInputValues(inputIndex);
292     inputIndex--;
293     SetInputValues(inputIndex);
294   }
295 }
296
297 void WriteInputFile(s)
298   char *s;
299 {
300   FILE *f;
301   int i,j,k;
302
303   f = fopen(s,"w");
304
305   fprintf(f, " &NAME\n");
306   fprintf(f, "    TMAX    = %10.4f\n", varValue[ 0]);
307   fprintf(f, "    LAMDA   = %10.4f\n", varValue[ 1]);
308   fprintf(f, "    TPRINT  = %10.4f\n", varValue[ 2]);
309   fprintf(f, "    NPUNCH  = %10.4f\n", varValue[ 3]);
310   fprintf(f, "    NBL     = %10.4f\n", varValue[ 4]);
311   fprintf(f, "    NDELX   = %10.4f\n", varValue[ 5]);
312   fprintf(f, "    TPI     = %10.4f\n", varValue[ 6]);
313   fprintf(f, "    PAM     = %10.4f\n", varValue[ 7]);
314   fprintf(f, "    UNIT    = %10.4f\n", varValue[ 8]);
315   fprintf(f, "    NINERT  = %10.4f\n", varValue[ 9]);
316   fprintf(f, "    NPNPXT  = %10.4f\n", varValue[10]);
317   fprintf(f, "    AT      = %10.4f\n", varValue[11]);
318   fprintf(f, "    XP      = %10.4f\n", varValue[12]);
319   fprintf(f, "    XIP     = %10.4f\n", varValue[13]);
320   fprintf(f, "    XG      = %10.4f\n", varValue[14]);
321   fprintf(f, "    XE      = %10.4f\n", varValue[15]);
322   fprintf(f, "    GAMA    = %10.4f\n", varValue[16]);
323   fprintf(f, "    TFREF   = %10.4f\n", varValue[17]);
324   fprintf(f, "    TIGN    = %10.4f\n", varValue[18]);
325   fprintf(f, "    RUFSUR  = %10.4f\n", varValue[19]);
326   fprintf(f, "    DDRG    = %10.4f\n", varValue[20]);
327   fprintf(f, "    DDHC    = %10.4f\n", varValue[21]);
328   fprintf(f, "    APIAT   = %10.4f\n", varValue[22]);
329   fprintf(f, "    AFTOS   = %10.4f\n", varValue[23]);
330   fprintf(f, "    FKPR    = %10.4f\n", varValue[24]);
331   fprintf(f, "    ROPR    = %10.4f\n", varValue[25]);
332   fprintf(f, "    CPR     = %10.4f\n", varValue[26]);
333   fprintf(f, "    TOREF   = %10.4f\n", varValue[27]);
334   fprintf(f, "    SIGP    = %10.4f\n", varValue[28]);
335   fprintf(f, "    TPScri  = %10.4f\n", varValue[29]);
336   fprintf(f, "    RREF    = %10.4f\n", varValue[30]);
337   fprintf(f, "    PREF    = %10.4f\n", varValue[31]);
338   fprintf(f, "    BREXP   = %10.4f\n", varValue[32]);
339   fprintf(f, "    EBC     = %10.4f\n", varValue[33]);
340   fprintf(f, "    EBEX    = %10.4f\n", varValue[34]);
341   fprintf(f, "    DE      = %10.4f\n", varValue[35]);
342   fprintf(f, "    CM      = %10.4f\n", varValue[36]);
343   fprintf(f, "    ALFAD   = %10.4f\n", varValue[37]);
344   fprintf(f, "    EROAT   = %10.4f\n", varValue[38]);
345   fprintf(f, "    EROEXP  = %10.4f\n", varValue[39]);
346   fprintf(f, "    NIGTAB  = %10.4f\n", varValue[40]);
347   fprintf(f, "    NAPDVX  = %10.4f\n", varValue[41]);
348   fprintf(f, "    NDATA   = %10.4f\n", varValue[42]);
349   fprintf(f, "    NPPR    = %10.4f\n", varValue[43]);
350   fprintf(f, "    PZONE   = %10.4f\n", varValue[44]);
351   fprintf(f, "    NRESRT  = %10.4f\n", varValue[45]);
352   fprintf(f, "    DELTAT  = %10.4f\n", varValue[46]);
353   fprintf(f, "    PCI     = %10.4f\n", varValue[47]);
354   fprintf(f, "    GAMAN   = %10.4f\n", varValue[48]);
355   fprintf(f, "    PISUBK  = %10.4f\n", varValue[49]);
356   fprintf(f, "    DELTTF  = %10.4f\n", varValue[50]);
357   fprintf(f, "    DFSDT   = %10.4f\n", varValue[51]);
358   fprintf(f, "    TIIN    = %10.4f\n", varValue[52]);
359   fprintf(f, "    FCRIT   = %10.4f\n", varValue[53]);

```

```

360     fprintf(f,"      POPEN  = %10.4f\n",varValue[54]);
361     fprintf(f,"      DELFAC = %10.4f\n",varValue[55]);
362     fprintf(f,"      CHC   = %10.4f\n",varValue[56]);
363     fprintf(f,"      BF1   = %10.4f\n",varValue[57]);
364     fprintf(f,"      DELTFF = %10.4f\n",varValue[58]);
365     fprintf(f," &END\n");
366
367     for (i=0;i<varValue[NIGTAB];i++)
368         fprintf(f,"%10.4f%10.4f\n",iTTable[j][0],iTTable[j][1]);
369
370     for (i=0;i<varValue[NAPDVX];i++) {
371         fprintf(f,"%10.4f%10.4f%10.4f%10.4f",
372             gTable[j][0],gTable[j][1],gTable[j][2],gTable[j][3]);
373         if (gTable[j][4]!=0) fprintf(f,"%10.4f",gTable[j][4]);
374         if (gTable[j][5]!=0) fprintf(f,"%10.4f",gTable[j][5]);
375         fprintf(f,"\n");
376     }
377     for (i=0;i<varValue[NDATA];i++)
378         fprintf(f,"%10.4f%10.4f\n",bTable[j][0],bTable[j][1]);
379     fclose(f);
380 }
381
382
383
384 */
1  /*
2  -----
3  xplot.c
4  -----
5  */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpWidgets.h"
11 #include "xpGeneral.h"
12 #include "xpMenus.h"
13 #include "xpCommands.h"
14 #include "xpInput.h"
15 #include "xpData.h"
16 #include "xpFormats.h"
17 #include "xpPlots.h"
18 #include "xpGeometry.h"
19
20 void main(argc,argv)
21     int argc; char **argv;
22 {
23     FILE *fi;
24     static XtActionsRec xplotActions[] = {
25         {"idle",Idle},
26         {"doKey",DoKey},
27         {"redrawPlot",RedrawPlot},
28         {"saveCommand",SaveCommand},
29         {"recallCommand",RecallCommand},
30         {"resizePlot",ResizePlot},
31         {"resizeGeometry",ResizeGeometry},
32         {"exposeGeometry",ExposeGeometry},
33         {"acceptInputEntry",AcceptInputEntry},
34     };
35     XtApplicationContext app_context;
36     int i;
37
38     fi = fopen("XPlot.dat","w");
39     topLevel = XtVaAppInitialize(&app_context,"XPlot",NULL,0,
40         &argc,argv,NULL,NULL);
41     XtAppAddActions(app_context,xplotActions,XtNumber(xplotActions));
42
43     CreateWidgets();
44     XtRealizeWidget(topLevel);

```

```

45     screen =XtScreen(topLevel);
46     display=XtDisplay(topLevel);
47
48     LoadFonts();
49     LoadColors();
50     InitData();
51     InitGCs();
52     for(i=0;i<MAXPLOTS;i++)
53         InitFormat(i);
54     LoadInputData();
55
56     SetFEValues(0);
57     SetStatusLabel();
58     ChangeMGLabels(0);
59
60     printf("\n");
61
62     XtInstallAccelerators(rootPane,cliText);
63     XtInstallAccelerators(cliLabel,cliText);
64
65     XtAppMainLoop(app_context);
66
67 }
68
69 /*
70  -----
71  xpMenus.c
72  -----
73 */
74
75 #include "xpIncs.h"
76 #include "xpDefs.h"
77
78 #include "xpwidgets.h"
79 #include "xpgeneral.h"
80 #include "xpcommands.h"
81 #include "xpdata.h"
82 #include "xpformats.h"
83 #include "xpplots.h"
84
85 /* $1: Global variables - (none) */
86
87 /* $2: Local variables - (none) */
88
89 /* $3: Global functions */
90
91 void MenuSelect();
92 void CommandSelect();
93 void VerbCallback();
94 void SFCallback();
95 void SFGetFile();
96 void SelectString();
97 void ShowHide2();
98 void LoadInput();
99 void FileCallback();
100 void ExecCallback();
101
102 /* $4: Local functions */
103 void SetStatusLabel();
104 void RunExec();
105
106 /* $5: Implementation ===== */
107
108 void MenuSelect(w,client_data,call_data)
109     Widget w; XtPointer client_data,call_data;
110 {
111     int menuData = (int) client_data;
112     int menuNum,itemNum;

```

```

45     char s[64],*t;
46     FILE *f;
47
48     menuNum = menuData/HIWORD;
49     itemNum = menuData&HIWORD;
50
51     switch(menuNum) {
52         case IDMMenuFile:
53             switch(itemNum) {
54                 case 0:
55                     ShowHide2(dummy,wFile,True,0);
56                     break;
57                 case 1:
58                     XtVaGetValues(menuText,XtNstring,&t,NULL);
59                     strcpy(s,t);
60                     f = fopen(s,"r");
61                     if (f!=NULL)
62                         XtVaSetValues(fileText,XtNstring,s,XtNtype,XawAsciiFile,NULL);
63                     break;
64                 case 2:
65                     ShowHide2(dummy,wFile,True,0);
66                     break;
67                 case 3:
68                     break;
69                 case 4:
70                     XtVaGetValues(menuText,XtNstring,&t,NULL);
71                     strcpy(s,t);
72                     f = fopen(s,"w");
73                     XtVaGetValues(fileText,XtNstring,&t,NULL);
74                     fprintf(f,"%s",t);
75                     fclose(f);
76                 case IDMFileQuit:
77                     exit(0);
78             }
79             break;
80         case IDMMenuInpu:
81             switch (itemNum) {
82                 case IDmInpuSet:
83                     SelectString(0);
84                     break;
85                 case IDmInpuLoad:
86                     LoadInput();
87                     break;
88                 case IDmInpuLoadSF:
89                     SelectString(0);
90                     LoadInput();
91                     break;
92                 case 4:
93                     ShowHide2(dummy,wInput,True,0);
94                     break;
95             }
96             break;
97         case IDMMenuExec:
98             switch (itemNum) {
99                 case IDmExecSetup:
100                     ShowHide2(dummy,wExecute,True,0);
101                     break;
102                 case IDmExecRun:
103                     RunExec();
104                     break;
105             }
106             break;
107         case IDMMenuData:
108             switch(itemNum) {
109                 case IDmDataSet:
110                     SelectString(1);
111                     break;
112                 case IDmDataLoad:
113                     LoadData(dsName);

```

```

114         break;
115     case IDmDataLoadSF:
116         SelectString(1);
117         LoadData(dsName);
118         break;
119     case IDmDataView:
120         ShowHide2(dummy,wData,True,0);
121         break;
122     case IDmDataPrint:
123         PrintData("ofile.dat");
124         break;
125     }
126     break;
127 case IDMenuForm:
128     switch(itemNum) {
129         case IDmFormSet:
130             SelectString(2);
131             break;
132         case IDmFormLoadSF:
133             SelectString(2);
134             break;
135     }
136     break;
137 case IDMenuPlot:
138     switch (itemNum) {
139         case IDmPlotSet:
140             SelectString(3);
141             break;
142         case IDmPlotInc:
143             IncDecPlot(True);
144             break;
145         case IDmPlotDec:
146             IncDecPlot(False);
147             break;
148     }
149     break;
150 case IDMenuWind:
151     switch (itemNum) {
152         case 0:
153             ShowHide2(dummy,w WindowManager,True,0);
154             break;
155         case 1:
156             ShowHide2(dummy,w File,True,0);
157             break;
158         case 2:
159             ShowHide2(dummy,w Input,True,0);
160             break;
161         case 3:
162             ShowHide2(dummy,w Execute,True,0);
163             break;
164         case 4:
165             ShowHide2(dummy,w Data,True,0);
166             break;
167         case 5:
168             ShowHide2(dummy,w Format,True,0);
169             break;
170         case 6:
171             ShowHide2(dummy,w FormatAxes,True,0);
172             break;
173         case 7:
174             ShowHide2(dummy,w FormatLines,True,0);
175             break;
176         case 8:
177             ShowHide2(dummy,w FormatCurveFits,True,0);
178             break;
179         case 9:
180             ShowHide2(dummy,w PlotManager,True,0);
181             break;
182         case 10:

```

```

183         ShowHide2(dummy,wMotorGeometry,True,0);
184         break;
185     case 11:
186         ShowHide2(dummy,wMotorConfiguration,True,0);
187         break;
188     }
189     break;
190 }
191 }
192 void CommandSelect(w,client_data,call_data)
193     Widget w; XtPointer client_data,call_data;
194 {
195     int commNum = (int) client_data;
196     char s[64];
197     static Boolean up[1] = {False};
198
199     switch (commNum)
200 {
201     case 0:
202         ShowHide2(dummy,wWindowManager,True,0);
203         break;
204     case 1:
205         ShowHide2(dummy,wFile,True,0);
206         break;
207     case 2:
208         ShowHide2(dummy,wInput,True,0);
209         break;
210     case 3:
211         ShowHide2(dummy,wExecute,True,0);
212         break;
213     case 4:
214         ShowHide2(dummy,wData,True,0);
215         break;
216     case 5:
217         ShowHide2(dummy,wFormat,True,0);
218         break;
219     case 6:
220         ShowHide2(dummy,wFormatAxes,True,0);
221         break;
222     case 7:
223         ShowHide2(dummy,wFormatLines,True,0);
224         break;
225     case 8:
226         ShowHide2(dummy,wFormatCurveFits,True,0);
227         break;
228     case 9:
229         ShowHide2(dummy,wPlotManager,True,0);
230         break;
231     case 10:
232         ShowHide2(dummy,wMotorGeometry,True,0);
233         break;
234     case 11:
235         ShowHide2(dummy,wMotorConfiguration,True,0);
236         break;
237     }
238 }
239 }
240 void ShowHide2(w,id,show,code)
241     Widget w; int id; Boolean show; int code;
242 {
243     char s[64];
244
245     if ((id>=0)&&(id<8)&&(plotExists[id])) {
246         if (show) {
247             windowUp[id]=True;
248             XtPopup(plotShell[id],XtGrabNone);
249         }
250     else {

```

```

252         windowUp[id]=False;
253         XtPopdown(plotShell[id]);
254     }
255 }
256 switch (id) {
257     case wWindowManager:
258         if (!windowUp[id]) {
259             strcpy(s,"Hide Window Manager");
260             XtPopup(commandShell,XtGrabNone);
261         }
262         else {
263             strcpy(s,"Show Window Manager");
264             XtPopdown(commandShell);
265         }
266         XtVaSetValues(c[0],XtNlabel,s,NULL);
267         XtVaSetValues(mEntry[6][0],XtNlabel,s,NULL);
268         windowUp[id]=!windowUp[id];
269         break;
270     case wFile:
271         if (!windowUp[id]) {
272             strcpy(s,"Hide File Editor");
273             XtPopup(fileShell,XtGrabNone);
274         }
275         else {
276             strcpy(s,"Show File Editor");
277             XtPopdown(fileShell);
278         }
279         XtVaSetValues(c[1],XtNlabel,s,NULL);
280         XtVaSetValues(mEntry[6][1],XtNlabel,s,NULL);
281         windowUp[id]=!windowUp[id];
282         break;
283     case wInput:
284         if (!windowUp[id]) {
285             strcpy(s,"Hide Input File Editor");
286             XtPopup(IEShell,XtGrabNone);
287         }
288         else {
289             strcpy(s,"Show Input File Editor");
290             XtPopdown(IEShell);
291         }
292         XtVaSetValues(c[2],XtNlabel,s,NULL);
293         XtVaSetValues(mEntry[6][2],XtNlabel,s,NULL);
294         windowUp[id]=!windowUp[id];
295         break;
296     case wExecute:
297         if (!windowUp[id]) {
298             strcpy(s,"Hide Execution Setup");
299             XtPopup(execShell,XtGrabNone);
300         }
301         else {
302             strcpy(s,"Show Execution Setup");
303             XtPopdown(execShell);
304         }
305         XtVaSetValues(c[3],XtNlabel,s,NULL);
306         XtVaSetValues(mEntry[6][3],XtNlabel,s,NULL);
307         windowUp[id]=!windowUp[id];
308         break;
309     case wData:
310         if (!windowUp[id]) {
311             strcpy(s,"Hide Current Dataset");
312             XtPopup(dataShell,XtGrabNone);
313         }
314         else {
315             strcpy(s,"Show Current Dataset");
316             XtPopdown(dataShell);
317         }
318         XtVaSetValues(c[4],XtNlabel,s,NULL);
319         XtVaSetValues(mEntry[6][4],XtNlabel,s,NULL);
320         windowUp[id]=!windowUp[id];

```

```

321         break;
322     case wFormat:
323         if (!windowUp[id]) {
324             strcpy(s, "Hide Format");
325             XtPopup(FEShell, XtGrabNone);
326         }
327         else {
328             strcpy(s, "Show Format");
329             XtPopdown(FEShell);
330         }
331         XtVaSetValues(c[5], XtNlabel, s, NULL);
332         XtVaSetValues(mEntry[6][5], XtNlabel, s, NULL);
333         windowUp[id]=!windowUp[id];
334         break;
335     case wFormatAxes:
336         break;
337     case wFormatLines:
338         if (!windowUp[id]) {
339             strcpy(s, "Hide Format Lines");
340             XtPopup(DAShell, XtGrabNone);
341         }
342         else {
343             strcpy(s, "Show Format Lines");
344             XtPopdown(DAShell);
345         }
346         XtVaSetValues(c[7], XtNlabel, s, NULL);
347         XtVaSetValues(mEntry[6][7], XtNlabel, s, NULL);
348         windowUp[id]=!windowUp[id];
349         break;
350     case wFormatCurveFits:
351         if (!windowUp[id]) {
352             strcpy(s, "Hide Format Curve Fits");
353             XtPopup(CFShell, XtGrabNone);
354         }
355         else {
356             strcpy(s, "Show Format Curve Fits");
357             XtPopdown(CFShell);
358         }
359         XtVaSetValues(c[8], XtNlabel, s, NULL);
360         XtVaSetValues(mEntry[6][8], XtNlabel, s, NULL);
361         windowUp[id]=!windowUp[id];
362         break;
363     case wPlotManager:
364         if (!windowUp[id]) {
365             strcpy(s, "Hide Plot Manager");
366             XtPopup(PMShell, XtGrabNone);
367         }
368         else {
369             strcpy(s, "Show Plot Manager");
370             XtPopdown(PMShell);
371         }
372         XtVaSetValues(c[9], XtNlabel, s, NULL);
373         XtVaSetValues(mEntry[6][9], XtNlabel, s, NULL);
374         windowUp[id]=!windowUp[id];
375         break;
376     case wMotorGeometry:
377         if (!windowUp[id]) {
378             strcpy(s, "Hide Motor Geometry");
379             XtPopup(MGShell, XtGrabNone);
380         }
381         else {
382             strcpy(s, "Show Motor Geometry");
383             XtPopdown(MGShell);
384         }
385         XtVaSetValues(c[10], XtNlabel, s, NULL);
386         XtVaSetValues(mEntry[6][10], XtNlabel, s, NULL);
387         windowUp[id]=!windowUp[id];
388         break;
389     case wMotorConfiguration:

```

```

390         if (!windowUp[id]) {
391             strcpy(s,"Hide Motor Configuration");
392             XtPopup(MGPictShell,XtGrabNone);
393         }
394         else {
395             strcpy(s,"Show Motor Configuration");
396             XtPopdown(MGPictShell);
397         }
398         XtVaSetValues(c[11],XtNlabel,s,NULL);
399         XtVaSetValues(mEntry[6][11],XtNlabel,s,NULL);
400         windowUp[id]=!windowUp[id];
401         break;
402     }
403 }
404
405 }
406
407 void VerbCallback(w,client_data,call_data)
408     Widget w; XtPointer client_data,call_data;
409 {
410     XawListReturnStruct *r = (XawListReturnStruct*) call_data;
411
412     strcpy(cli,(*r).string);
413     XtVaSetValues(cliText,XtNstring,cli,NULL);
414     XawTextSetInsertionPoint(cliText,strlen(cli));
415 }
416
417
418 /*
419 void SFCallback(w,client_data,call_data)
420     Widget w; XtPointer client_data,call_data;
421 {
422     int action = (int) client_data;
423     char filename[32],s[64];
424     String f;
425
426     f = XawDialogGetValueString(sfDialog);
427     strcpy(filename,f);
428     XtPopdown(sfGetFileShell);
429
430     switch (action) {
431         case SFCommCancel:
432             break;
433         case SFCommLoad:
434             sprintf(s,"The filename selected was: %s\n",filename);
435             break;
436         case SFCommSave:
437             break;
438     }
439 }
440
441 void SFGetFile()
442 {
443     XtPopup(sfGetFileShell,XtGrabNone);
444 }
445 */
446
447 void SelectString(code)
448     int code;
449 {
450     char s[64],*ss;
451     int i;
452
453     XtVaGetValues(menuText,XtNstring,&ss,NULL);
454     strncpy(s,ss,63);
455     s[63]='\0';
456     if(s[0]!='\0') {
457         switch (code) {
458             case 0:

```

```

459         strcpy(inName,s);
460         break;
461     case 1:
462         strcpy(dsName,s);
463         break;
464     case 2:
465         strcpy(pfName,s);
466         break;
467     case 3:
468         i=atoi(s);
469         if((i>=0)&&(i<=7)&&(plotExists[i])) {
470             activePlot=i;
471             sprintf(plName,"%d",i);
472         }
473         break;
474     }
475     SetStatusLabel();
476 }
477 }
478
479 void SetStatusLabel()
480 {
481     char s[256];
482
483     strcpy(s,"I: <"); 
484     strcat(s,inName);
485     strcat(s,> " D: <"); 
486     strcat(s,dsName);
487     strcat(s,> " F: <"); 
488     strcat(s,pfName);
489     strcat(s,> " P: <"); 
490     strcat(s,plName);
491     strcat(s,>"); 
492
493     XtVaSetValues(statusLabel,XtNlabel,s,NULL);
494 }
495
496 void FileCallback(w,client_data,call_data)
497     Widget w; XtPointer client_data,call_data;
498 {
499     int n = (int) client_data;
500     FILE *f;
501     char s[128],*t;
502
503     XtVaGetValues(fileEntry[3],XtNstring,&t,NULL);
504     strcpy(s,t);
505     XtVaGetValues(fileText,XtNstring,&t,NULL);
506
507     switch (n) {
508         case 0:
509             ShowHide2(dummy,wFile,False,0);
510             break;
511         case 1:
512             f = fopen(s,"w");
513             fprintf(f,"%s",t);
514             fclose(f);
515             break;
516         case 2:
517             f = fopen(s,"r");
518             if (f!=NULL)
519                 XtVaSetValues(fileText,XtNstring,s,XtNtype,XawAsciiFile,NULL);
520             break;
521     }
522
523 }
524
525
526 void LoadInput()
527 {

```

```

528     FILE *fi;
529     char s[128],*t;
530
531     XtVaGetValues(fileEntry[3],XtNstring,&t,NULL);
532     strcpy(s,t);
533
534     fi = fopen(s,"r");
535     if (fi!=NULL) {
536         XtVaSetValues(fileText,XtNstring,s,XtNtype,XawAsciiFile,NULL);
537     }
538 }
540
541 void ExecCallback(w,client_data,call_data)
542     Widget w; XtPointer client_data,call_data;
543 {
544     int n = (int) client_data;
545
546     switch (n) {
547         case 0:
548             ShowHide2(dummy,wExecute,False,0);
549             break;
550         case 1:
551             break;
552         case 2:
553             RunExec();
554             break;
555     }
556 }
557
558 void RunExec()
559 {
560     char s[128],t[128],*u;
561
562     XtVaGetValues(execItem[1][0],XtNstring,&u,NULL);
563     strcpy(s,u);
564     XtVaGetValues(execItem[1][1],XtNstring,&u,NULL);
565     strcpy(t,u);
566     strcat(s,t);
567     strcat(s," & ");
568     XtVaGetValues(execItem[1][2],XtNstring,&u,NULL);
569     strcpy(t,u);
570     if (strlen(t)!=0) {
571         strcat(s," < ");
572         strcat(s,t);
573     }
574     XtVaGetValues(execItem[1][3],XtNstring,&u,NULL);
575     strcpy(t,u);
576     if (strlen(t)!=0) {
577         strcat(s," > ");
578         strcat(s,t);
579     }
580     system(s);
581 }
582 */
1  -----
2  ----- xpPlots.c
3  -----
4  */
5
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 #include "xpWidgets.h"
11 #include "xpGeneral.h"
12 #include "xpData.h"
13 #include "xpFormats.h"
14

```

```

15  /* $1: Global Variables */
16
17  unsigned long lineColor[64];
18  unsigned long curvColor[32];
19
20  int activePlot=-1;
21  Boolean plotExists[MAXPLOTS];
22  Boolean useCol[MAXPLOTS][2*MAXCOLS];
23  Boolean planar[MAXPLOTS]=(True,True,True,True,True,True,True);
24
25  /* $2: Local variables */
26
27  XSegment Grid[MAXPLOTS][50];
28  XPoint pt[MAXPLOTS][2*MAXCOLS][MAXPLANES];
29  XPoint pt2[MAXPLOTS][2*MAXCOLS][128];
30
31  char xLabels[MAXPLOTS][24][16];
32  char yLabels[MAXPLOTS][24][16];
33  char zLabels[MAXPLOTS][24][16];
34
35  Dimension oldWidth[MAXPLOTS] ={640,640,640,640,640,640,640,640};
36  Dimension oldHeight[MAXPLOTS]=(480,480,480,480,480,480,480,480);
37
38  /* $3: Global functions */
39
40  void MakePlot();
41  void KillPlot();
42  void ActivatePlot();
43  void RedrawPlot();
44  void RedrawPlot2();
45  void ResizePlot();
46  void PMActiveCallback();
47  void PMCreateCallback();
48  void PMShowHideCallback();
49  void PMPlanarCallback();
50  void PMActionCallback();
51  void IncDecPlot();
52
53  /* $4: Local functions */
54
55  void RecreatePlot();
56
57  void SetupGrid();
58  void SetupAxisLabels();
59  void SetupPoints();
60
61  void DrawGrid();
62  void DrawAxisLabels();
63  void DrawPlotLabels();
64  void DrawLegend();
65  void DrawPoints();
66
67  void SmoothAxisValues();
68
69  void GESCP();
70  void poly();
71  void spline();
72
73  /* $5: Implementation ===== */
74
75  void PMActiveCallback(w,client_data,call_data)
76    Widget w; XtPointer client_data,call_data;
77  {
78    int n = (int) XawToggleGetCurrent(PMitem[0][0]);
79
80    if ((n>=1)&&(n<=8))
81      activePlot=n-1;
82  }
83

```

```

84 void PMCreateCallback(w,client_data,call_data)
85   Widget w; XtPointer client_data,call_data;
86 {
87   int p = (int) client_data;
88
89   if ((!plotExists[p])&&(dataLoaded)) {
90     XtVaSetValues(PMitem[0][p],XtNsensitive,True,NULL);
91     XtVaSetValues(PMitem[1][p],XtNlabel,"Destroy",NULL);
92     XtVaSetValues(PMitem[2][p],XtNsensitive,True,XtNlabel,"Hide",NULL);
93     MakePlot(p);
94     ShowHide2(plotShell[p],p,True,0);
95   }
96   else if (plotExists[p]) {
97     KillPlot(p);
98     XtVaSetValues(PMitem[0][p],XtNsensitive,False,NULL);
99     XtVaSetValues(PMitem[1][p],XtNlabel,"Create",NULL);
100    XtVaSetValues(PMitem[2][p],XtNsensitive,False,XtNlabel,"Show",NULL);
101  }
102 }
103
104 void PMShowHideCallback(w,client_data,call_data)
105   Widget w; XtPointer client_data,call_data;
106 {
107   int p = (int) client_data;
108
109   if (windowUp[p]) {
110     XtVaSetValues(PMitem[2][p],XtNlabel,"Show",NULL);
111     ShowHide2(plotShell[p],p,False,0);
112   }
113   else {
114     XtVaSetValues(PMitem[2][p],XtNlabel,"Hide",NULL);
115     ShowHide2(plotShell[p],p,True,0);
116   }
117 }
118
119 void PMPlanarCallback(w,client_data,call_data)
120   Widget w; XtPointer client_data,call_data;
121 {
122   int which = (int) client_data;
123
124   if ((activePlot>=0)&&(activePlot<MAXPLOTS)) {
125     if (planar[activePlot]) {
126       planar[activePlot] = False;
127       XtVaSetValues(PMitem[3][0],XtNlabel,"Plane Plot",NULL);
128     }
129     else {
130       planar[activePlot]=True;
131       XtVaSetValues(PMitem[3][0],XtNlabel,"Space Plot",NULL);
132     }
133     CopyData(activePlot);
134     StatData(activePlot);
135     /* PrintPlotData(activePlot); */
136     SetupGrid(activePlot);
137     SetupAxisLabels(activePlot);
138     SetupPoints(activePlot);
139     RedrawPlot2(activePlot);
140   }
141 }
142
143 void PMActionCallback(w,client_data,call_data)
144   Widget w; XtPointer client_data,call_data;
145 {
146   int commNum = (int) client_data;
147
148   switch (commNum) {
149     case 1:
150       IncDecPlot(True);
151       break;
152     case 2:

```

```

153         IncDecPlot(False);
154         break;
155     case 3:
156         if ((activePlot>=0)&&(activePlot<=7)) {
157             if (planar[activePlot])
158                 while (currentPlane[activePlot]<(numPlanes-1))
159                     IncDecPlot(True);
160             else
161                 while (currentRow[activePlot]<(numRows-1))
162                     IncDecPlot(True);
163         }
164         break;
165     case 4:
166         if ((activePlot>=0)&&(activePlot<=7)) {
167             if (planar[activePlot])
168                 while (currentPlane[activePlot]>0)
169                     IncDecPlot(False);
170             else
171                 while (currentRow[activePlot]>0)
172                     IncDecPlot(False);
173         }
174         break;
175     case 5:
176         ShowHide2(dummy,wFormat,True,0);
177         break;
178     case 6:
179         ShowHide2(dummy,wPlotManager,True,0);
180         break;
181     }
182 }
183
184 void MakePlot(p)
185     int p;
186 {
187     plotExists[p]=True;
188
189     if(dataLoaded) {
190         CopyData(p);
191         StatInd();
192         StatData(p);
193         /* PrintPlotData(p); */
194         SetupGrid(p);
195         SetupAxisLabels(p);
196         SetupPoints(p);
197         ActivatePlot(p);
198     }
199     else
200         printf("No dataset currently loaded\n");
201 }
202
203 void KillPlot(p)
204     int p;
205 {
206     ShowHide2(dummy,wPlot0+p,False,0);
207     windowUp[wPlot0+p]=False;
208     plotExists[p]=False;
209     if (p==activePlot) {
210         XawToggleUnsetCurrent(PMitem[0][0]);
211         activePlot=-1;
212     }
213 }
214
215 void ActivatePlot(p)
216     int p;
217 {
218     if((p>=0)&&(p<MAXPLOTS)&&(plotExists[p])) {
219         activePlot = p;
220         XawToggleSetCurrent(PMitem[0][0],(caddr_t) p+1);
221

```

```

222     if (planar[activePlot])
223         XtVaSetValues(PMitem[3][0], XtNlabel, "Space Plot", NULL);
224     else
225         XtVaSetValues(PMitem[3][0], XtNlabel, "Plane Plot", NULL);
226     }
227 }
228 void RedrawPlot(w, event, params, numParams)
229     Widget w; XEvent *event; String *params; Cardinal *numParams;
230 {
231     XExposeEvent *myevent = (XExposeEvent*) event;
232     Window wind=XtWindow(w);
233     static int i=0;
234     int p = atoi(params[0]);
235
236     if ((*myevent).count==0) {
237         i++;
238         XClearWindow(display,wind);
239         DrawGrid(p);
240         DrawAxisLabels(p);
241         DrawPlotLabels(p);
242         DrawLegend(p);
243         DrawPoints(p);
244
245         printf(":RedrawPlot(%d) - %d\n",p,i);
246     }
247 }
248 void RedrawPlot2(p)
249     int p;
250 {
251     Window wind=XtWindow(plot[p]);
252
253     XClearWindow(display,wind);
254     DrawGrid(p);
255     DrawAxisLabels(p);
256     DrawPlotLabels(p);
257     DrawLegend(p);
258     DrawPoints(p);
259
260 }
261 void ResizePlot(widget, event, params, numParams)
262     Widget widget; XEvent *event; String *params; Cardinal *numParams;
263 {
264     Dimension w,h;
265     int p = atoi(params[0]);
266
267     XtVaGetValues(plotShell[p], XtNwidth, &w, XtNheight, &h, NULL);
268     /* printf(":ResizePlot(%d) - (%d,%d)\n",p,w,h); */
269
270     f[p].width = w;
271     f[p].height = h;
272
273     if ((w!=oldWidth[p])||(h!=oldHeight[p])) {
274         RecalcFormat(p);
275         SetupGrid(p);
276         SetupPoints(p);
277     }
278     if ((w<=oldWidth[p])&&(h<=oldHeight[p]))
279         RedrawPlot2(p);
280
281     oldWidth[p] =w;
282     oldHeight[p]=h;
283
284 }
285 void IncDecPlot(inc)
286     Boolean inc;
287 {
288     Boolean changed=False;
289
290

```

```

291     if ((activePlot>=0)&&(activePlot<MAXPLOTS))
292     if (planar[activePlot]) {
293         if ((inc)&&(currentPlane[activePlot]<(numPlanes-1))) {
294             currentPlane[activePlot]++;
295             changed=True;
296         }
297         if ((!inc)&&(currentPlane[activePlot]>0)) {
298             currentPlane[activePlot]--;
299             changed=True;
300         }
301     }
302     else {
303         if ((inc)&&(currentRow[activePlot]<(numRows-1))) {
304             currentRow[activePlot]++;
305             changed=True;
306         }
307         if ((!inc)&&(currentRow[activePlot]>0)) {
308             currentRow[activePlot]--;
309             changed=True;
310         }
311     }
312     if (changed) {
313         CopyData(activePlot);
314         StatData(activePlot);
315         /* PrintPlotData(activePlot); */
316         SetupGrid(activePlot);
317         SetupAxisLabels(activePlot);
318         SetupPoints(activePlot);
319         RedrawPlot2(activePlot);
320     }
321 }
322 }
323 void RecreatePlot(p)
324     int p;
325 {
326     CopyData(p);
327     StatData(p);
328     PrintPlotData(p);
329     SetupGrid(p);
330     SetupAxisLabels(p);
331     SetupPoints(p);
332     RedrawPlot2(p);
333 }
334
335 void SetupGrid(p)
336     int p;
337 {
338     int dx=f[p].xInc,dy=f[p].yInc;
339     int numx=f[p].xNum,numy=f[p].yNum;
340     int lengthx=f[p].xLength,lengthy=f[p].yLength;
341     int bL=f[p].borderL,bR=f[p].borderR;
342     int bT=f[p].borderT,bB=f[p].borderB;
343     int i;
344
345     for (i=0;i<=numy;i++) {
346         Grid[p][i].x1 = bL-f[p].yTick;
347         Grid[p][i].y1 = bT+i*dy;
348         Grid[p][i].x2 = bL+lengthx;
349         Grid[p][i].y2 = bT+i*dy;
350     }
351     for (i=0;i<=numx;i++) {
352         Grid[p][i+1+numy].x1 = bL+i*dx;
353         Grid[p][i+1+numy].y1 = bT;
354         Grid[p][i+1+numy].x2 = bL+i*dx;
355         Grid[p][i+1+numy].y2 = bT+lengthy+f[p].xTick;
356     }
357 }
358 }
359

```

```

360 void DrawGrid(p)
361   int p;
362 {
363   Window wind=XtWindow(plot[p]);
364   XSetForeground(display,gc[0],rColor[Black].pixel);
365   XDrawSegments(display,wind,gc[0],Grid[p],f[p].nGrid);
366 }
367
368 void SetupAxisLabels(p)
369   int p;
370 {
371   float temp,minx,miny,maxx,maxy,minz,maxz,rangex,rangey,rangez;
372   int numx=f[p].xNum,numy=f[p].yNum;
373   int i;
374   char s[16],t[16];
375
376   sprintf(s,"%%%sf",f[p].xFormat);
377   sprintf(t,"%%%sf",f[p].yFormat);
378
379   if (!f[p].opt[oAutoScale]) {
380     minx=f[p].xMin;
381     maxx=f[p].xMax;
382     rangex=f[p].xMax-f[p].xMin;
383     minz=f[p].xMin;
384     maxz=f[p].xMax;
385     rangez=f[p].xMax-f[p].xMin;
386     miny=f[p].yMin;
387     maxy=f[p].yMax;
388     rangey=f[p].yMax-f[p].yMin;
389   }
390   else {
391     if (planar[p]) {
392       minx=xMin;
393       maxx=xMax;
394       rangex=xRange;
395     }
396     else {
397       minz=zMin;
398       maxz=zMax;
399       rangez=zRange;
400     }
401     miny=yMin[p];
402     maxy=yMax[p];
403     rangey=yRange[p];
404   }
405
406   if (planar[p]) {
407     for(i=0;i<=numx;i++) {
408       temp=minx+i*rangex/numx;
409       sprintf(xLabels[p][i],s,temp);
410     }
411     for(i=0;i<=numy;i++) {
412       temp=miny+i*rangey/numy;
413       sprintf(yLabels[p][i],t,temp);
414     }
415   }
416   else {
417     for(i=0;i<=numx;i++) {
418       temp=minz+i*rangez/numx;
419       sprintf(xLabels[p][i],s,temp);
420     }
421     for(i=0;i<=numy;i++) {
422       temp=miny+i*rangey/numy;
423       sprintf(yLabels[p][i],t,temp);
424     }
425   }
426 }
427
428 }

```

```

429 void DrawAxisLabels(p)
430   int p;
431 {
432   Window wind=XtWindow(plot[p]);
433   int dx=f[p].xInc,dy=f[p].yInc;
434   int maxx=f[p].xNum,maxy=f[p].yNum;
435   int i,w,x,y;
436
437   x=f[p].borderL;
438   y=f[p].borderT+f[p].yLength+f[p].xTick+12;
439
440   XSetForeground(display,gc[0],rColor[Black].pixel);
441   for(i=0;i<=maxx;i++) {
442     w=XTextWidth(fs6x10,xLabels[p][i],strlen(xLabels[p][i]))/2;
443     XDrawString(display,wind,gc[0],x+i*dx-w,y,
444     xLabels[p][i],strlen(xLabels[p][i]));
445   }
446   x=f[p].borderL-f[p].yTick-6;
447   y=f[p].borderT+f[p].yLength;
448   for(i=0;i<=maxy;i++) {
449     w=XTextWidth(fs6x10,yLabels[p][i],strlen(yLabels[p][i]));
450     XDrawString(display,wind,gc[0],x-w,y-i*dy,
451     yLabels[p][i],strlen(yLabels[p][i]));
452   }
453 }
454
455 void DrawPlotLabels(p)
456   int p;
457 {
458   Window wind=XtWindow(plot[p]);
459   char t[64];
460   int w,n,h,v;
461
462   XSetForeground(display,gc[0],rColor[Black].pixel);
463   strcpy(t,f[p].title);
464   w=XTextWidth(fsHelv18b,t,strlen(t));
465   h=f[p].borderL+(f[p].xLength-w)/2;
466   v=f[p].borderT-8;
467   XDrawString(display,wind,gc[2],h,v,t,strlen(t));
468   if (planar[p])
469     sprintf(t,"Distance (inch) [Time = %6.0f : %3d/%3d]",
470     z[currentPlane[p]],currentPlane[p]+1,numPlanes);
471   else
472     sprintf(t,"Time (msec) [Dist = %6.0f : %3d/%3d]",
473     x[currentRow[p]],currentRow[p]+1,numRows);
474   w = XTextWidth(fs6x10,t,strlen(t));
475   h=f[p].borderL+(f[p].xLength-w)/2;
476   v=f[p].borderT+f[p].yLength+30;
477   XDrawString(display,wind,gc[0],h,v,t,strlen(t));
478 }
479
480 void DrawLegend(p)
481   int p;
482 {
483   Window wind=XtWindow(plot[p]);
484   int j,k=-1;
485   XSegment Legend[1];
486   int len=15,xDist=100,yDist=14,x,y;
487   int test;
488
489   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
490   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
491   for(j=0;j<test;j++)
492     if (useCol[p][j]) {
493       k++;
494       x=f[p].borderL+(k%4)*xDist;
495       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
496       Legend[0].x1=x;
497       Legend[0].y1=y;
498       Legend[0].x2=x+15;
499       Legend[0].y2=y;
500     }
501   XDrawSegments(display,wind,gc[0],Legend,1);
502 }
503
504 void DrawPlot(p)
505   int p;
506 {
507   Window wind=XtWindow(plot[p]);
508   int i,j,k;
509   XSegment Line[1];
510   int len=15,xDist=100,yDist=14,x,y;
511   int test;
512
513   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
514   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
515   for(j=0;j<test;j++)
516     if (useCol[p][j]) {
517       k++;
518       x=f[p].borderL+(k%4)*xDist;
519       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
520       Line[0].x1=x;
521       Line[0].y1=y;
522       Line[0].x2=x+15;
523       Line[0].y2=y;
524     }
525   XDrawSegments(display,wind,gc[0],Line,1);
526 }
527
528 void DrawPlot(p)
529   int p;
530 {
531   Window wind=XtWindow(plot[p]);
532   int i,j,k;
533   XSegment Line[1];
534   int len=15,xDist=100,yDist=14,x,y;
535   int test;
536
537   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
538   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
539   for(j=0;j<test;j++)
540     if (useCol[p][j]) {
541       k++;
542       x=f[p].borderL+(k%4)*xDist;
543       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
544       Line[0].x1=x;
545       Line[0].y1=y;
546       Line[0].x2=x+15;
547       Line[0].y2=y;
548     }
549   XDrawSegments(display,wind,gc[0],Line,1);
550 }
551
552 void DrawPlot(p)
553   int p;
554 {
555   Window wind=XtWindow(plot[p]);
556   int i,j,k;
557   XSegment Line[1];
558   int len=15,xDist=100,yDist=14,x,y;
559   int test;
560
561   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
562   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
563   for(j=0;j<test;j++)
564     if (useCol[p][j]) {
565       k++;
566       x=f[p].borderL+(k%4)*xDist;
567       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
568       Line[0].x1=x;
569       Line[0].y1=y;
570       Line[0].x2=x+15;
571       Line[0].y2=y;
572     }
573   XDrawSegments(display,wind,gc[0],Line,1);
574 }
575
576 void DrawPlot(p)
577   int p;
578 {
579   Window wind=XtWindow(plot[p]);
580   int i,j,k;
581   XSegment Line[1];
582   int len=15,xDist=100,yDist=14,x,y;
583   int test;
584
585   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
586   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
587   for(j=0;j<test;j++)
588     if (useCol[p][j]) {
589       k++;
590       x=f[p].borderL+(k%4)*xDist;
591       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
592       Line[0].x1=x;
593       Line[0].y1=y;
594       Line[0].x2=x+15;
595       Line[0].y2=y;
596     }
597   XDrawSegments(display,wind,gc[0],Line,1);
598 }
599
600 void DrawPlot(p)
601   int p;
602 {
603   Window wind=XtWindow(plot[p]);
604   int i,j,k;
605   XSegment Line[1];
606   int len=15,xDist=100,yDist=14,x,y;
607   int test;
608
609   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
610   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
611   for(j=0;j<test;j++)
612     if (useCol[p][j]) {
613       k++;
614       x=f[p].borderL+(k%4)*xDist;
615       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
616       Line[0].x1=x;
617       Line[0].y1=y;
618       Line[0].x2=x+15;
619       Line[0].y2=y;
620     }
621   XDrawSegments(display,wind,gc[0],Line,1);
622 }
623
624 void DrawPlot(p)
625   int p;
626 {
627   Window wind=XtWindow(plot[p]);
628   int i,j,k;
629   XSegment Line[1];
630   int len=15,xDist=100,yDist=14,x,y;
631   int test;
632
633   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
634   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
635   for(j=0;j<test;j++)
636     if (useCol[p][j]) {
637       k++;
638       x=f[p].borderL+(k%4)*xDist;
639       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
640       Line[0].x1=x;
641       Line[0].y1=y;
642       Line[0].x2=x+15;
643       Line[0].y2=y;
644     }
645   XDrawSegments(display,wind,gc[0],Line,1);
646 }
647
648 void DrawPlot(p)
649   int p;
650 {
651   Window wind=XtWindow(plot[p]);
652   int i,j,k;
653   XSegment Line[1];
654   int len=15,xDist=100,yDist=14,x,y;
655   int test;
656
657   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
658   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
659   for(j=0;j<test;j++)
660     if (useCol[p][j]) {
661       k++;
662       x=f[p].borderL+(k%4)*xDist;
663       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
664       Line[0].x1=x;
665       Line[0].y1=y;
666       Line[0].x2=x+15;
667       Line[0].y2=y;
668     }
669   XDrawSegments(display,wind,gc[0],Line,1);
670 }
671
672 void DrawPlot(p)
673   int p;
674 {
675   Window wind=XtWindow(plot[p]);
676   int i,j,k;
677   XSegment Line[1];
678   int len=15,xDist=100,yDist=14,x,y;
679   int test;
680
681   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
682   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
683   for(j=0;j<test;j++)
684     if (useCol[p][j]) {
685       k++;
686       x=f[p].borderL+(k%4)*xDist;
687       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
688       Line[0].x1=x;
689       Line[0].y1=y;
690       Line[0].x2=x+15;
691       Line[0].y2=y;
692     }
693   XDrawSegments(display,wind,gc[0],Line,1);
694 }
695
696 void DrawPlot(p)
697   int p;
698 {
699   Window wind=XtWindow(plot[p]);
700   int i,j,k;
701   XSegment Line[1];
702   int len=15,xDist=100,yDist=14,x,y;
703   int test;
704
705   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
706   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
707   for(j=0;j<test;j++)
708     if (useCol[p][j]) {
709       k++;
710       x=f[p].borderL+(k%4)*xDist;
711       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
712       Line[0].x1=x;
713       Line[0].y1=y;
714       Line[0].x2=x+15;
715       Line[0].y2=y;
716     }
717   XDrawSegments(display,wind,gc[0],Line,1);
718 }
719
720 void DrawPlot(p)
721   int p;
722 {
723   Window wind=XtWindow(plot[p]);
724   int i,j,k;
725   XSegment Line[1];
726   int len=15,xDist=100,yDist=14,x,y;
727   int test;
728
729   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
730   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
731   for(j=0;j<test;j++)
732     if (useCol[p][j]) {
733       k++;
734       x=f[p].borderL+(k%4)*xDist;
735       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
736       Line[0].x1=x;
737       Line[0].y1=y;
738       Line[0].x2=x+15;
739       Line[0].y2=y;
740     }
741   XDrawSegments(display,wind,gc[0],Line,1);
742 }
743
744 void DrawPlot(p)
745   int p;
746 {
747   Window wind=XtWindow(plot[p]);
748   int i,j,k;
749   XSegment Line[1];
750   int len=15,xDist=100,yDist=14,x,y;
751   int test;
752
753   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
754   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
755   for(j=0;j<test;j++)
756     if (useCol[p][j]) {
757       k++;
758       x=f[p].borderL+(k%4)*xDist;
759       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
760       Line[0].x1=x;
761       Line[0].y1=y;
762       Line[0].x2=x+15;
763       Line[0].y2=y;
764     }
765   XDrawSegments(display,wind,gc[0],Line,1);
766 }
767
768 void DrawPlot(p)
769   int p;
770 {
771   Window wind=XtWindow(plot[p]);
772   int i,j,k;
773   XSegment Line[1];
774   int len=15,xDist=100,yDist=14,x,y;
775   int test;
776
777   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
778   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
779   for(j=0;j<test;j++)
780     if (useCol[p][j]) {
781       k++;
782       x=f[p].borderL+(k%4)*xDist;
783       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
784       Line[0].x1=x;
785       Line[0].y1=y;
786       Line[0].x2=x+15;
787       Line[0].y2=y;
788     }
789   XDrawSegments(display,wind,gc[0],Line,1);
790 }
791
792 void DrawPlot(p)
793   int p;
794 {
795   Window wind=XtWindow(plot[p]);
796   int i,j,k;
797   XSegment Line[1];
798   int len=15,xDist=100,yDist=14,x,y;
799   int test;
800
801   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
802   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
803   for(j=0;j<test;j++)
804     if (useCol[p][j]) {
805       k++;
806       x=f[p].borderL+(k%4)*xDist;
807       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
808       Line[0].x1=x;
809       Line[0].y1=y;
810       Line[0].x2=x+15;
811       Line[0].y2=y;
812     }
813   XDrawSegments(display,wind,gc[0],Line,1);
814 }
815
816 void DrawPlot(p)
817   int p;
818 {
819   Window wind=XtWindow(plot[p]);
820   int i,j,k;
821   XSegment Line[1];
822   int len=15,xDist=100,yDist=14,x,y;
823   int test;
824
825   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
826   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
827   for(j=0;j<test;j++)
828     if (useCol[p][j]) {
829       k++;
830       x=f[p].borderL+(k%4)*xDist;
831       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
832       Line[0].x1=x;
833       Line[0].y1=y;
834       Line[0].x2=x+15;
835       Line[0].y2=y;
836     }
837   XDrawSegments(display,wind,gc[0],Line,1);
838 }
839
840 void DrawPlot(p)
841   int p;
842 {
843   Window wind=XtWindow(plot[p]);
844   int i,j,k;
845   XSegment Line[1];
846   int len=15,xDist=100,yDist=14,x,y;
847   int test;
848
849   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
850   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
851   for(j=0;j<test;j++)
852     if (useCol[p][j]) {
853       k++;
854       x=f[p].borderL+(k%4)*xDist;
855       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
856       Line[0].x1=x;
857       Line[0].y1=y;
858       Line[0].x2=x+15;
859       Line[0].y2=y;
860     }
861   XDrawSegments(display,wind,gc[0],Line,1);
862 }
863
864 void DrawPlot(p)
865   int p;
866 {
867   Window wind=XtWindow(plot[p]);
868   int i,j,k;
869   XSegment Line[1];
870   int len=15,xDist=100,yDist=14,x,y;
871   int test;
872
873   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
874   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
875   for(j=0;j<test;j++)
876     if (useCol[p][j]) {
877       k++;
878       x=f[p].borderL+(k%4)*xDist;
879       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
880       Line[0].x1=x;
881       Line[0].y1=y;
882       Line[0].x2=x+15;
883       Line[0].y2=y;
884     }
885   XDrawSegments(display,wind,gc[0],Line,1);
886 }
887
888 void DrawPlot(p)
889   int p;
890 {
891   Window wind=XtWindow(plot[p]);
892   int i,j,k;
893   XSegment Line[1];
894   int len=15,xDist=100,yDist=14,x,y;
895   int test;
896
897   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
898   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
899   for(j=0;j<test;j++)
900     if (useCol[p][j]) {
901       k++;
902       x=f[p].borderL+(k%4)*xDist;
903       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
904       Line[0].x1=x;
905       Line[0].y1=y;
906       Line[0].x2=x+15;
907       Line[0].y2=y;
908     }
909   XDrawSegments(display,wind,gc[0],Line,1);
910 }
911
912 void DrawPlot(p)
913   int p;
914 {
915   Window wind=XtWindow(plot[p]);
916   int i,j,k;
917   XSegment Line[1];
918   int len=15,xDist=100,yDist=14,x,y;
919   int test;
920
921   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
922   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
923   for(j=0;j<test;j++)
924     if (useCol[p][j]) {
925       k++;
926       x=f[p].borderL+(k%4)*xDist;
927       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
928       Line[0].x1=x;
929       Line[0].y1=y;
930       Line[0].x2=x+15;
931       Line[0].y2=y;
932     }
933   XDrawSegments(display,wind,gc[0],Line,1);
934 }
935
936 void DrawPlot(p)
937   int p;
938 {
939   Window wind=XtWindow(plot[p]);
940   int i,j,k;
941   XSegment Line[1];
942   int len=15,xDist=100,yDist=14,x,y;
943   int test;
944
945   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
946   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
947   for(j=0;j<test;j++)
948     if (useCol[p][j]) {
949       k++;
950       x=f[p].borderL+(k%4)*xDist;
951       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
952       Line[0].x1=x;
953       Line[0].y1=y;
954       Line[0].x2=x+15;
955       Line[0].y2=y;
956     }
957   XDrawSegments(display,wind,gc[0],Line,1);
958 }
959
960 void DrawPlot(p)
961   int p;
962 {
963   Window wind=XtWindow(plot[p]);
964   int i,j,k;
965   XSegment Line[1];
966   int len=15,xDist=100,yDist=14,x,y;
967   int test;
968
969   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
970   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
971   for(j=0;j<test;j++)
972     if (useCol[p][j]) {
973       k++;
974       x=f[p].borderL+(k%4)*xDist;
975       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
976       Line[0].x1=x;
977       Line[0].y1=y;
978       Line[0].x2=x+15;
979       Line[0].y2=y;
980     }
981   XDrawSegments(display,wind,gc[0],Line,1);
982 }
983
984 void DrawPlot(p)
985   int p;
986 {
987   Window wind=XtWindow(plot[p]);
988   int i,j,k;
989   XSegment Line[1];
990   int len=15,xDist=100,yDist=14,x,y;
991   int test;
992
993   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
994   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
995   for(j=0;j<test;j++)
996     if (useCol[p][j]) {
997       k++;
998       x=f[p].borderL+(k%4)*xDist;
999       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1000      Line[0].x1=x;
1001      Line[0].y1=y;
1002      Line[0].x2=x+15;
1003      Line[0].y2=y;
1004    }
1005   XDrawSegments(display,wind,gc[0],Line,1);
1006 }
1007
1008 void DrawPlot(p)
1009   int p;
1010 {
1011   Window wind=XtWindow(plot[p]);
1012   int i,j,k;
1013   XSegment Line[1];
1014   int len=15,xDist=100,yDist=14,x,y;
1015   int test;
1016
1017   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1018   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1019   for(j=0;j<test;j++)
1020     if (useCol[p][j]) {
1021       k++;
1022       x=f[p].borderL+(k%4)*xDist;
1023       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1024       Line[0].x1=x;
1025       Line[0].y1=y;
1026       Line[0].x2=x+15;
1027       Line[0].y2=y;
1028     }
1029   XDrawSegments(display,wind,gc[0],Line,1);
1030 }
1031
1032 void DrawPlot(p)
1033   int p;
1034 {
1035   Window wind=XtWindow(plot[p]);
1036   int i,j,k;
1037   XSegment Line[1];
1038   int len=15,xDist=100,yDist=14,x,y;
1039   int test;
1040
1041   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1042   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1043   for(j=0;j<test;j++)
1044     if (useCol[p][j]) {
1045       k++;
1046       x=f[p].borderL+(k%4)*xDist;
1047       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1048       Line[0].x1=x;
1049       Line[0].y1=y;
1050       Line[0].x2=x+15;
1051       Line[0].y2=y;
1052     }
1053   XDrawSegments(display,wind,gc[0],Line,1);
1054 }
1055
1056 void DrawPlot(p)
1057   int p;
1058 {
1059   Window wind=XtWindow(plot[p]);
1060   int i,j,k;
1061   XSegment Line[1];
1062   int len=15,xDist=100,yDist=14,x,y;
1063   int test;
1064
1065   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1066   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1067   for(j=0;j<test;j++)
1068     if (useCol[p][j]) {
1069       k++;
1070       x=f[p].borderL+(k%4)*xDist;
1071       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1072       Line[0].x1=x;
1073       Line[0].y1=y;
1074       Line[0].x2=x+15;
1075       Line[0].y2=y;
1076     }
1077   XDrawSegments(display,wind,gc[0],Line,1);
1078 }
1079
1080 void DrawPlot(p)
1081   int p;
1082 {
1083   Window wind=XtWindow(plot[p]);
1084   int i,j,k;
1085   XSegment Line[1];
1086   int len=15,xDist=100,yDist=14,x,y;
1087   int test;
1088
1089   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1090   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1091   for(j=0;j<test;j++)
1092     if (useCol[p][j]) {
1093       k++;
1094       x=f[p].borderL+(k%4)*xDist;
1095       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1096       Line[0].x1=x;
1097       Line[0].y1=y;
1098       Line[0].x2=x+15;
1099       Line[0].y2=y;
1100     }
1101   XDrawSegments(display,wind,gc[0],Line,1);
1102 }
1103
1104 void DrawPlot(p)
1105   int p;
1106 {
1107   Window wind=XtWindow(plot[p]);
1108   int i,j,k;
1109   XSegment Line[1];
1110   int len=15,xDist=100,yDist=14,x,y;
1111   int test;
1112
1113   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1114   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1115   for(j=0;j<test;j++)
1116     if (useCol[p][j]) {
1117       k++;
1118       x=f[p].borderL+(k%4)*xDist;
1119       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1120       Line[0].x1=x;
1121       Line[0].y1=y;
1122       Line[0].x2=x+15;
1123       Line[0].y2=y;
1124     }
1125   XDrawSegments(display,wind,gc[0],Line,1);
1126 }
1127
1128 void DrawPlot(p)
1129   int p;
1130 {
1131   Window wind=XtWindow(plot[p]);
1132   int i,j,k;
1133   XSegment Line[1];
1134   int len=15,xDist=100,yDist=14,x,y;
1135   int test;
1136
1137   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1138   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1139   for(j=0;j<test;j++)
1140     if (useCol[p][j]) {
1141       k++;
1142       x=f[p].borderL+(k%4)*xDist;
1143       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1144       Line[0].x1=x;
1145       Line[0].y1=y;
1146       Line[0].x2=x+15;
1147       Line[0].y2=y;
1148     }
1149   XDrawSegments(display,wind,gc[0],Line,1);
1150 }
1151
1152 void DrawPlot(p)
1153   int p;
1154 {
1155   Window wind=XtWindow(plot[p]);
1156   int i,j,k;
1157   XSegment Line[1];
1158   int len=15,xDist=100,yDist=14,x,y;
1159   int test;
1160
1161   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1162   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1163   for(j=0;j<test;j++)
1164     if (useCol[p][j]) {
1165       k++;
1166       x=f[p].borderL+(k%4)*xDist;
1167       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1168       Line[0].x1=x;
1169       Line[0].y1=y;
1170       Line[0].x2=x+15;
1171       Line[0].y2=y;
1172     }
1173   XDrawSegments(display,wind,gc[0],Line,1);
1174 }
1175
1176 void DrawPlot(p)
1177   int p;
1178 {
1179   Window wind=XtWindow(plot[p]);
1180   int i,j,k;
1181   XSegment Line[1];
1182   int len=15,xDist=100,yDist=14,x,y;
1183   int test;
1184
1185   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1186   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1187   for(j=0;j<test;j++)
1188     if (useCol[p][j]) {
1189       k++;
1190       x=f[p].borderL+(k%4)*xDist;
1191       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1192       Line[0].x1=x;
1193       Line[0].y1=y;
1194       Line[0].x2=x+15;
1195       Line[0].y2=y;
1196     }
1197   XDrawSegments(display,wind,gc[0],Line,1);
1198 }
1199
1200 void DrawPlot(p)
1201   int p;
1202 {
1203   Window wind=XtWindow(plot[p]);
1204   int i,j,k;
1205   XSegment Line[1];
1206   int len=15,xDist=100,yDist=14,x,y;
1207   int test;
1208
1209   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1210   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1211   for(j=0;j<test;j++)
1212     if (useCol[p][j]) {
1213       k++;
1214       x=f[p].borderL+(k%4)*xDist;
1215       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1216       Line[0].x1=x;
1217       Line[0].y1=y;
1218       Line[0].x2=x+15;
1219       Line[0].y2=y;
1220     }
1221   XDrawSegments(display,wind,gc[0],Line,1);
1222 }
1223
1224 void DrawPlot(p)
1225   int p;
1226 {
1227   Window wind=XtWindow(plot[p]);
1228   int i,j,k;
1229   XSegment Line[1];
1230   int len=15,xDist=100,yDist=14,x,y;
1231   int test;
1232
1233   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1234   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1235   for(j=0;j<test;j++)
1236     if (useCol[p][j]) {
1237       k++;
1238       x=f[p].borderL+(k%4)*xDist;
1239       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1240       Line[0].x1=x;
1241       Line[0].y1=y;
1242       Line[0].x2=x+15;
1243       Line[0].y2=y;
1244     }
1245   XDrawSegments(display,wind,gc[0],Line,1);
1246 }
1247
1248 void DrawPlot(p)
1249   int p;
1250 {
1251   Window wind=XtWindow(plot[p]);
1252   int i,j,k;
1253   XSegment Line[1];
1254   int len=15,xDist=100,yDist=14,x,y;
1255   int test;
1256
1257   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1258   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1259   for(j=0;j<test;j++)
1260     if (useCol[p][j]) {
1261       k++;
1262       x=f[p].borderL+(k%4)*xDist;
1263       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1264       Line[0].x1=x;
1265       Line[0].y1=y;
1266       Line[0].x2=x+15;
1267       Line[0].y2=y;
1268     }
1269   XDrawSegments(display,wind,gc[0],Line,1);
1270 }
1271
1272 void DrawPlot(p)
1273   int p;
1274 {
1275   Window wind=XtWindow(plot[p]);
1276   int i,j,k;
1277   XSegment Line[1];
1278   int len=15,xDist=100,yDist=14,x,y;
1279   int test;
1280
1281   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1282   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1283   for(j=0;j<test;j++)
1284     if (useCol[p][j]) {
1285       k++;
1286       x=f[p].borderL+(k%4)*xDist;
1287       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1288       Line[0].x1=x;
1289       Line[0].y1=y;
1290       Line[0].x2=x+15;
1291       Line[0].y2=y;
1292     }
1293   XDrawSegments(display,wind,gc[0],Line,1);
1294 }
1295
1296 void DrawPlot(p)
1297   int p;
1298 {
1299   Window wind=XtWindow(plot[p]);
1300   int i,j,k;
1301   XSegment Line[1];
1302   int len=15,xDist=100,yDist=14,x,y;
1303   int test;
1304
1305   test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
1306   XSetLineAttributes(display,gc[0],8,LineSolid,CapButt,JoinRound);
1307   for(j=0;j<test;j++)
1308     if (useCol[p][j]) {
1309       k++;
1310       x=f[p].borderL+(k%4)*xDist;
1311       y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
1312       Line[0].x1=x;
1313       Line[0].y1=y;
1314       Line[0].x2=x+15;
1315       Line[0].y2=y;
1316     }
1317   XDrawSegments(display,wind,gc[0],Line,1);
1318 }
1319
1320 void DrawPlot(p)
1321   int p;
1322 {
1323   Window wind=XtWindow(plot[p]);
1324   int i,j,k;
1325   XSegment Line[1];
1326   int len=15,xDist=100,yDist=14
```

```

498     Legend[0].y1=y-4;
499     Legend[0].x2=x+len;
500     Legend[0].y2=y-4;
501     XSetForeground(display,gc[0],rColor[Black].pixel);
502     XDrawString(display,wind,gc[0],x+len+5,y,
503         columnNames[j],strlen(columnNames[j]));
504     XSetForeground(display,gc[0],lineColor[f[p].color[j]]);
505     XDrawSegments(display,wind,gc[0],Legend,1);
506   }
507   XSetForeground(display,gc[0],rColor[Black].pixel);
508   XSetLineAttributes(display,gc[0],0,LineSolid,CapButt,JoinRound);
509 }
510
511 void SetupPoints(p)
512 {
513   int bL=f[p].borderL,bR=f[p].borderR;
514   int bT=f[p].borderT,bB=f[p].borderB;
515   float minx=xMin,miny=yMin[p],minz=zMin;
516   float rangex=xRange,rangey=yRange[p],rangez=zRange;
517   int lengthx=f[p].xLength,lengthy=f[p].yLength;
518   int i,j;
519
520   if (!f[p].opt[oAutoScale]) {
521     minx=f[p].xMin;
522     rangex=f[p].xMax-f[p].xMin;
523     minz=f[p].xMin;
524     rangez=f[p].xMax-f[p].xMin;
525     miny=f[p].yMin;
526     rangey=f[p].yMax-f[p].yMin;
527   }
528   else {
529     if (planar[p]) {
530       minx=xMin;
531       rangex=xRange;
532     }
533     else {
534       minz=zMin;
535       rangez=zRange;
536     }
537     miny=yMin[p];
538     rangey=yRange[p];
539   }
540
541   if (planar[p]) {
542     for(j=0;j<numCols;j++) {
543       if (useCol[p][j])
544         for(i=0;i<numRows;i++) {
545           pt[p][j][i].x=bL+((x[i]-minx)/rangex)*lengthx;
546           pt[p][j][i].y=bT+lengthy-((plotData[p][j][i]-miny)/
547             rangey)*lengthy;
548         }
549     }
550   }
551   else {
552     for(j=0;j<(2*MAXCOLS);j++) {
553       if (useCol[p][j])
554         for(i=0;i<numPlanes;i++) {
555           pt[p][j][i].x=bL+((z[i]-minz)/rangez)*lengthx;
556           pt[p][j][i].y=bT+lengthy-((plotData[p][j][i]-miny)/
557             rangey)*lengthy;
558         }
559     }
560   }
561 }
562 }
563
564 void DrawPoints(p)
565 {

```

```

567     Window wind=XtWindow(plot[p]);
568     static char dashPattern[] = {16,8};
569     int j,test;
570     XRectangle r;
571     int bL=f[p].borderL,bR=f[p].borderR;
572     int bT=f[p].borderT,bB=f[p].borderB;
573     int lengthx=f[p].xLength,lengthy=f[p].yLength;
574     XSetLineAttributes(display,gc[0],4,LineOnOffDash,CapRound,JoinRound);
575     XSetDashes(display,gc[0],0,dashPattern,2);
576     r.x=bL-2; r.y=bT-2; r.width=lengthx+4; r.height=lengthy+4;
577     XSetClipRectangles(display,gc[0],0,0,r,1,Unsorted);
578
579
580     test=(planar[p] ? MAXCOLUMNS : 2*MAXCOLUMNS);
581     for(j=0;j<test;j++)
582     {
583         if (useCol[p][j]) {
584             if (f[p].numDashes[j]>0) {
585                 XSetLineAttributes(display,gc[0],4,LineOnOffDash,CapRound,JoinRound);
586                 XSetDashes(display,gc[0],0,f[p].dashes[j],f[p].numDashes[j]);
587             }
588             else
589                 XSetLineAttributes(display,gc[0],4,LineSolid,CapRound,JoinRound);
590             XSetForeground(display,gc[0],lineColor[f[p].color[j]]);
591             if (planar[p])
592                 XDrawLines(display,wind,gc[0],pt[p][j],numRows,CoordModeOrigin);
593             else
594                 XDrawLines(display,wind,gc[0],pt[p][j],numPlanes,CoordModeOrigin);
595         }
596         XSetForeground(display,gc[0],rColor[Black].pixel);
597         XSetLineAttributes(display,gc[0],0,LineSolid,CapButt,JoinRound);
598         XSetClipMask(display,gc[0],None);
599     }
600 }
601 void SmoothAxisValues(vMin,vMax)
602 {
603     float vMin,vMax;
604     {
605         float vRange;
606         int a,a1,a2,b,b1,b2;
607
608         vRange = vMax-vMin;
609
610         a = log10(vRange);
611         a1= a-1;
612         a2= a-2;
613
614         b = vRange/a;
615
616         printf("%d",b);
617     }
618 }
619 /*
620 -----
621     The following routines apply only to curve-fitting (not implemented)
622 -----
623 */
624
625 void GESCP(n,a,b)
626 {
627     int n; float a[10][10],b[10];
628     {
629         int i,j,k,p,nrow[10],ncopy;
630         float max,m,sum,s[10];
631
632         for (i=0;i<n;i++) {
633             max = -1.0E+10;
634             for (j=0;j<n;j++)
635                 if (fabs(a[i][j]) >max) max = fabs(a[i][j]);

```

```

636     s[i]=max;
637     nrow[i]=i;
638 }
639 for (i=0;i<(n-1);i++) {
640     max=-1.0E+10;
641     for (j=i;j<n;j++) {
642         if ((fabs(a[nrow[j]][i])/s[nrow[j]]) > max) {
643             max = fabs(a[nrow[j]][i])/s[nrow[j]];
644             p=j;
645         }
646         if (a[nrow[p]][i]==0.0) printf("No unique solution exists\n");
647         if (nrow[i]!=nrow[p]) {
648             ncopy=nrow[i];
649             nrow[i]=nrow[p];
650             nrow[p]=ncopy;
651         }
652         for (j=(i+1);j<n;j++) {
653             m = a[nrow[j]][i]/a[nrow[i]][i];
654             for (k=0;k<(n+1);k++)
655                 a[nrow[j]][k]=a[nrow[j]][k]-m*a[nrow[i]][k];
656         }
657     }
658     if (a[nrow[n-1]][n-1]==0.0) printf("No unique solution exists\n");
659     b[n-1]=a[nrow[n-1]][n]/a[nrow[n-1]][n-1];
660     for (i=(n-2);i>=0;i--) {
661         sum=0;
662         for (j=i+1;j<n;j++)
663             sum=sum+a[nrow[i]][j]*b[j];
664         b[i] = (a[nrow[i]][n]-sum)/a[nrow[i]][i];
665     }
666 }
667
668 void poly(n,m,x,y)
669   int n,m; float x[],y[];
670 {
671   int i,j,k;
672   float a[10][10],b[10],p[256],e[256],sum;
673   for (i=0;i<n;i++) {
674     for (j=0;j<n;j++) {
675       sum=0;
676       for (k=0;k<m;k++)
677         sum=sum+((i+j)==0 ? 1.0 : (float) pow((double) x[k],(double) i+j)
);
678       if ((i+j)==0) sum=m;
679       a[i][j]=sum;
680     }
681   }
682   for (i=0;i<n;i++) {
683     sum=0;
684     for (k=0;k<m;k++)
685       sum=sum+y[k]*((i==0 ? 1.0 : (float) pow((double) x[k],(double) i) );
686     a[i][n]=sum;
687   }
688   GESCP(n,a,b);
689   for (i=0;i<n;i++)
690     printf("x[%d] = %12.6f\n",i,b[i]);
691   for (i=0;i<m;i++) {
692     sum=0;
693     for (k=0;k<n;k++)
694       sum=sum+b[k]*((k==0 ? 1.0 : (float) pow((double) x[i],(double) k) );
695     p[i]=sum;
696     e[i]=y[i]-p[i];
697     printf("%12.6f %12.6f%12.6f%12.6f\n",x[i],y[i],p[i],e[i]);
698   }
699   sum=0;
700   for (i=0;i<m;i++)
701     sum+=(float) pow((double) e[i],(double) 2);
702   printf("\n\nTotal error = %12.6f\n",sum);
703 }

```

```

704 void spline(n,x,y)
705     int n; float x[],y[];
706 {
707     int i,j;
708     float a[256],b[256],c[256],d[256],
709         h[256],alpha[256],l[256],u[256],z[256];
710
711     for (i=0;i<=n;i++)
712         a[i]=y[i];
713     for (i=0;i<n;i++)
714         h[i]=x[i+1]-x[i];
715     for (i=1;i<n;i++)
716         alpha[i]=3*(a[i+1]*h[i-1]-a[i]*(x[i+1]-x[i-1])+a[i-1]*h[i])/
717             (h[i-1]*h[i]);
718     l[0]=1;
719     u[0]=0;
720     z[0]=0;
721     for (i=1;i<n;i++) {
722         l[i]=2*(x[i+1]-x[i-1])-h[i-1]*u[i-1];
723         u[i]=h[i]/l[i];
724         z[i]=(alpha[i]-h[i-1]*z[i-1])/l[i];
725     }
726     l[n]=1.0;
727     z[n]=0.0;
728     c[n]=0.0;
729     for (j=(n-1);j>=0;j--) {
730         c[j]=z[j]-u[j]*c[j+1];
731         b[j]=(a[j+1]-a[j])/h[j]-h[j]*(c[j+1]+2*c[j])/3;
732         d[j]=(c[j+1]-c[j])/(3*h[j]);
733     }
734 }
735 }
736
1 /*
2 -----
3     xpWidgets.c
4 -----
5 */
6
7 #include "xpIncs.h"
8 #include "xpDefs.h"
9
10 /* The following files have to be included for callback routines */
11
12 #include "xpmenus.h"
13 #include "xpcommands.h"
14 #include "xpinput.h"
15 #include "xpformats.h"
16 #include "xpplots.h"
17 #include "xpgeometry.h"
18
19 /* $1: Global variables */
20
21 Widget topLevel;
22
23 Widget rootPane;
24 Widget menuPane,menuText;
25 Widget statusLabel;
26 Widget commPane,verbViewport,verbList,commViewport,commList;
27 Widget cliPane,cliLabel,cliText;
28
29 Widget mMenuButton[NUMMenus],mSimpleMenu[NUMMenus];
30 Widget mEntry[NUMMenus][MAXENTRIES];
31
32 Widget commandShell,commandBox,c[NUMCommands];
33
34 Widget PMShell,PMForm,PMBox[4],PMTop[4],PMitem[4][8];
35 Widget infoShell,infoCore;
36

```



```

175  /* _A pane to hold all of the menu bar entries */
176  menuPane = XtVaCreateManagedWidget("menuPane", panedWidgetClass, rootPane,
177      XtNorientation, XtorientHorizontal,
178      XtNshowGrip, False,
179      NULL);
180
181  /* _Generate the menu buttons, placing them one after another */
182  for(i=0;i<NUMMenus;i++)
183      mMenuButton[i] = XtVaCreateManagedWidget(
184          mMMenuButtonNames[i], menuButtonWidgetClass, menuPane,
185          XtNborderWidth, 0,
186          XtNshowGrip, False,
187          XtNmenuName, mMsimpleMenuNames[i],
188          NULL);
189  for(j=0;j<NUMMenus;j++) {
190      mMsimpleMenu[j] = XtVaCreatePopupShell(
191          mMsimpleMenuNames[j], simpleMenuWidgetClass, mMenuButton[j],
192          NULL);
193      for (i=0;i<numMenuEntries[j];i++) {
194          if (mEntryNames[j][i][0]=='/')
195              mEntry[j][i] = XtVaCreateManagedWidget(
196                  mEntryNames[j][i], smeLineObjectClass, mMsimpleMenu[j],
197                  NULL);
198          else {
199              mEntry[j][i] = XtVaCreateManagedWidget(
200                  mEntryNames[j][i], smeBSBObjectClass, mMsimpleMenu[j],
201                  NULL);
202              XtAddCallback(mEntry[j][i], XtNcallback, MenuSelect,
203                  (XtPointer) (j*HIWORD+i));
204          }
205      }
206  }
207  menuText = XtVaCreateManagedWidget(
208      "menuText", asciiTextWidgetClass, menuPane,
209      XtNborderWidth, 0,
210      XtNshowGrip, False,
211      XtNeditType, XawtextEdit,
212      NULL);
213
214  statusLabel = XtVaCreateManagedWidget(
215      "statusLabel", labelWidgetClass, rootPane,
216      XtNwidth, 600,
217      XtNshowGrip, False,
218      XtNskipAdjust, True,
219      NULL);
220
221  commPane = XtVaCreateManagedWidget(
222      "commPane", panedWidgetClass, rootPane,
223      XtNorientation, XtorientHorizontal,
224      XtNshowGrip, False,
225      NULL);
226  verbViewport = XtVaCreateManagedWidget(
227      "verbViewport", viewportWidgetClass, commPane,
228      XtNmin, 100,
229      XtNallowVert, True,
230      XtNforceBars, True,
231      XtNshowGrip, False,
232      NULL);
233  verbList = XtVaCreateManagedWidget(
234      "verbList", listWidgetClass, verbViewport,
235      XtNverticalList, True,
236      XtNcolumnSpacing, 0,
237      XtNrowSpacing, 2,
238      XtNdefaultColumns, 1,
239      XtNforceColumns, True,
240      XtNlist, verbs,
241      NULL);
242  XtAddCallback(verbList, XtNcallback, VerbCallback, 0);
243  commViewport = XtVaCreateManagedWidget(

```

```

244     "commViewport",viewportWidgetClass,commPane,
245     XtNallowVert,True,
246     XtNforceBars,True,
247     XtNshowGrip,False,
248     NULL);
249     commList = XtVaCreateManagedWidget(
250         "commList",listWidgetClass,commViewport,
251         XtNverticalList,True,
252         XtNcolumnSpacing,0,
253         XtNdefaultColumns,1,
254         XtNforceColumns,True,
255         XtNlist,commElements,
256         XtNnumberStrings,15,
257         NULL);
258     XtAddCallback(commList,XtNcallback,ListCallback,0);
259
260     cliPane = XtVaCreateManagedWidget(
261         "cliPane",panedWidgetClass,rootPane,
262         XtNskipAdjust,True,
263         XtNorientation,XtorientHorizontal,
264         XtNshowGrip,False,
265         NULL);
266     cliLabel = XtVaCreateManagedWidget(
267         "cliLabel",labelWidgetClass,cliPane,
268         XtNlabel,"",
269         XtNshowGrip,False,
270         XtNjustify,XtJustifyLeft,
271         XtNmin,100,
272         XtNresize,False,
273         NULL);
274     cliText = XtVaCreateManagedWidget(
275         "cliText",asciiTextWidgetClass,cliPane,
276         XtNeditType,XawtextEdit,
277         NULL);
278
279 /* _Create the widgets used to make the command palette work */
280     commandShell = XtVaCreatePopupShell(
281         "commandShell",transientShellWidgetClass,topLevel,
282         XtNtitle,"Window Manager",
283         NULL);
284     commandBox = XtVaCreateManagedWidget("commandBox",boxWidgetClass,
285         commandShell, NULL);
286     for (i=0;i<12;i++) {
287         sprintf(s,"c%02d",i);
288         c[i] = XtVaCreateManagedWidget(
289             s,commandWidgetClass,commandBox,
290             XtNborderWidth,0,
291             XtNwidth,200,
292             XtNresize,False,
293             XtNjustify,XtJustifyLeft,
294             NULL);
295         XtAddCallback(c[i],XtNcallback,CommandSelect,(XtPointer) i);
296     };
297
298 /* _Create the widgets used to make the command line log work */
299 /*
300     logShell = XtVaCreatePopupShell(
301         "logShell",transientShellWidgetClass,topLevel,
302         XtNtitle,"Command Line Log",
303         NULL);
304     logText = XtVaCreateManagedWidget(
305         "logText",asciiTextWidgetClass,logShell,
306         XtNwidth,480,
307         XtNheight,240,
308         XtNeditType,XawtextAppend,
309         XtNscrollVertical,XawtextScrollAlways,
310         XtNshowGrip,False,
311         NULL);
312 */

```

```

313  /* _Create the widgets used to view plots (up to MAXPLOTS=8) */
314  for(i=0;i<MAXPLOTS;i++) {
315      char name[64],label[64];
316      sprintf(name , "plotShell_%d",i);
317      sprintf(label, "xplot: Plot #%d",i);
318      plotShell[i] = XtVaCreatePopupShell(
319          name,transientShellWidgetClass,topLevel,
320          XtNminWidth,320,
321          XtNminHeight,240,
322          XtNsaveUnder, False,
323          XtNtitle, label,
324          NULL);
325      sprintf(name, "plot%d",i);
326      plot[i] = XtVaCreateManagedWidget(
327          name,widgetClass,plotShell[i],
328          XtNwidth,640,
329          XtNheight,480,
330          NULL);
331  }
332  */
333  /* _Create the widgets used to view the current dataset */
334  dataShell = XtVaCreatePopupShell(
335      "dataShell",transientShellWidgetClass,topLevel,
336      XtNtitle, "Dataset",
337      NULL);
338  dataText = XtVaCreateManagedWidget(
339      "dataText",asciiTextWidgetClass,dataShell,
340      XtNwidth, 480,
341      XtNheight,240,
342      XtNeditType,XawtextRead,
343      XtNtype,XawAsciiFile,
344      XtNscrollVertical,XawtextScrollAlways,
345      XtNstring,"XPlot.dat",
346      NULL);
347  */
348  /* _Create the widgets used for a standard get|put file dialog */
349  /*
350  */
351  sfGetFileShell = XtVaCreatePopupShell(
352      "sfGetFileShell",transientShellWidgetClass,topLevel,
353      XtNtitle, "Filename Dialog",
354      NULL);
355  sfDialog = XtVaCreateManagedWidget(
356      "sfDialog",dialogWidgetClass,sfGetFileShell,
357      XtNlabel, "Load Dataset (Named):",
358      XtNvalue, "",
359      NULL);
360  sfCancel = XtVaCreateManagedWidget(
361      "sfCancel",commandWidgetClass,sfDialog,
362      XtNlabel, "Cancel",
363      NULL);
364  sfLoad = XtVaCreateManagedWidget(
365      "sfLoad",commandWidgetClass,sfDialog,
366      XtNlabel, "Load",
367      NULL);
368  sfSave = XtVaCreateManagedWidget(
369      "sfSave",commandWidgetClass,sfDialog,
370      XtNlabel, "Save",
371      NULL);
372  XtAddCallback(sfCancel,XtNcallback,SFCallback,(XtPointer) 0);
373  XtAddCallback(sfLoad ,XtNcallback,SFCallback,(XtPointer) 1);
374  XtAddCallback(sfSave ,XtNcallback,SFCallback,(XtPointer) 2);
375  */
376  /*
377  */
378  IEShell = XtVaCreatePopupShell(
379      "IEShell",transientShellWidgetClass,topLevel,
380      XtNtitle, "Input File Editor",
381      NULL);

```

```

382     IEFForm = XtVaCreateManagedWidget(
383         "IEForm", formWidgetClass, IEShell,
384         NULL);
385     for (i=0;i<8;i++) {
386         sprintf(s,"IEBox_%d",i);
387         IEBox[i] = XtVaCreateManagedWidget(
388             s,boxWidgetClass,IEForm,
389             XtNhspace,1,
390             XtNvspace,1,
391             XtNorientation,XtorientHorizontal,
392             NULL);
393         sprintf(s,"IETop_%d",i);
394         IETop[i] = XtVaCreateManagedWidget(
395             s,labelWidgetClass,IEBox[i],
396             XtNresize,False,
397             NULL);
398     }
399     for (i=0;i<6;i++) {
400         sprintf(s,"IEItem_%d",i);
401         if ( (i<3) || (i==5) )
402             IEItem[i][0] = XtVaCreateManagedWidget(
403                 s,asciiTextWidgetClass,IEBox[i],
404                 XtNdisplayCaret,False,
405                 NULL);
406         else
407             IEItem[i][0] = XtVaCreateManagedWidget(
408                 s,asciiTextWidgetClass,IEBox[i],
409                 XtNeditType,XawtextEdit,
410                 NULL);
411     }
412     for (i=0;i<6;i++) {
413         sprintf(s,"IEItem_6_%d",i);
414         IEItem[6][i] = XtVaCreateManagedWidget(
415             s,commandWidgetClass,IEBox[6],
416             NULL);
417         XtAddCallback(IEItem[6][i],XtNcallback,IECallback,(XtPointer) i);
418     }
419     IEViewport = XtVaCreateManagedWidget(
420         "IEViewport",viewportWidgetClass,IEBox[7],
421         XtNallowVert,True,
422         XtNforceBars,True,
423         XtNtop,XtChainTop,XtNfromHoriz,IEBox[0],
424         NULL);
425     IEList = XtVaCreateManagedWidget(
426         "IEList",listWidgetClass,IEViewport,
427         XtNverticalList,True,
428         XtNcolumnSpacing,0,
429         XtNrowSpacing,2,
430         XtNdefaultColumns,1,
431         XtNforceColumns,True,
432         XtNlist,IEentries,
433         NULL);
434     XtAddCallback(IEList,XtNcallback,IEListCallback,0);
435
436
437
438
439     XtVaSetValues(IEBox[0],XtNleft,XtChainLeft,XtNtop,XtChainTop,NULL);
440     for (i=1;i<7;i++)
441         XtVaSetValues(IEBox[i],XtNfromVert,IEBox[i-1],
442             XtNleft,XtChainLeft,NULL);
443
444     XtVaSetValues(IEBox[7],XtNfromHoriz,IEBox[0],XtNtop,XtChainTop,
445             XtNorientation,XtorientVertical,NULL);
446
447     /* _Create the widgets used in the dialog for the colors and hashes */
448
449     DAShell = XtVaCreatePopupShell(
450         "DAShell", transientShellWidgetClass, topLevel,

```

```

451      XtNtitle, "Data Attributes Editor",
452      NULL);
453      DAForm = XtVaCreateManagedWidget(
454          "DAForm", formWidgetClass, DAShell,
455          NULL);
456  /*
457      DALabel[0] = XtVaCreateManagedWidget(
458          "DALabel_0", labelWidgetClass, DAForm,
459          XtNfromHoriz, DABox[0],
460          NULL);
461      DALabel[1] = XtVaCreateManagedWidget(
462          "DALabel_1", labelWidgetClass, DAForm,
463          XtNfromHoriz, DALabel[0],
464          NULL);
465  */
466  for (i=0;i<5;i++) {
467      sprintf(s, "DABox_%d", i);
468      DABox[i] = XtVaCreateManagedWidget(
469          s, boxWidgetClass, DAForm,
470          XtNhSpace, 1,
471          XtNvSpace, 1,
472          XtNorientation, XtorientVertical,
473          XtNtop, XtChainTop,
474          NULL);
475      sprintf(s, "DATop_%d", i);
476      DATop[i] = XtVaCreateManagedWidget(
477          s, labelWidgetClass, DABox[i],
478          XtNresize, False,
479          NULL);
480  }
481  for (i=1;i<5;i++)
482      XtVaSetValues(DABox[i], XtNfromHoriz, DABox[i-1], NULL);
483
484  for (i=0;i<16;i++) {
485      sprintf(s, "DAIItem_0_%02d", i);
486      DAIItem[0][i] = XtVaCreateManagedWidget(
487          s, labelWidgetClass, DABox[0],
488          XtNresize, False,
489          XtNlabel, " ",
490          NULL);
491  }
492  for (j=1;j<5;j++) {
493      for (i=0;i<16;i++) {
494          sprintf(s, "DAIItem_%d_%02d", j, i);
495          DAIItem[j][i] = XtVaCreateManagedWidget(
496              s, asciiTextWidgetClass, DABox[j],
497              XtNeditType, XawtextEdit,
498              NULL);
499      }
500  /*
501  * _Create the widgets used in the dialog to edit formats */
502
503  FEShell = XtVaCreatePopupShell(
504      "FEShell", transientShellWidgetClass, topLevel,
505      XtNtitle, "Format Editor",
506      NULL);
507  FEForm = XtVaCreateManagedWidget(
508      "FEForm", formWidgetClass, FEShell,
509      NULL);
510
511  for (i=0;i<10;i++) {
512      sprintf(s, "FEBox_%d", i);
513      FEBox[i] = XtVaCreateManagedWidget(
514          s, boxWidgetClass, FEForm,
515          XtNhSpace, 1,
516          XtNvSpace, 1,
517          NULL);
518      sprintf(s, "FETop_%d", i);
519      FETop[i] = XtVaCreateManagedWidget(

```

```

520         s,labelWidgetClass,FEBox[i],
521         XtNresize,False,
522         NULL);
523     }
524     XtVaSetValues(FEBox[9],XtNleft,XtChainLeft,XtNtop,XtChainTop,
525         XtNorientation,XtorientHorizontal,NULL);
526     XtVaSetValues(FEBox[0],XtNleft,XtChainLeft,XtNfromVert,FEBox[9],
527         XtNorientation,XtorientHorizontal,NULL);
528     XtVaSetValues(FEBox[1],XtNleft,XtChainLeft,NULL);
529     XtVaSetValues(FEBox[5],XtNleft,XtChainLeft,NULL);
530     for (i=1;i<5;i++)
531         XtVaSetValues(FEBox[i],XtNfromVert,FEBox[0],
532             XtNfromHoriz,FEBox[i-1],NULL);
533     for (i=5;i<9;i++)
534         XtVaSetValues(FEBox[i],XtNfromVert,FEBox[1],
535             XtNfromHoriz,FEBox[i-1],NULL);
536     XtVaSetValues(FEBox[1],XtNfromHoriz,NULL,NULL);
537     XtVaSetValues(FEBox[5],XtNfromHoriz,NULL,NULL);
538
539
540     FEItem[9][0] = XtVaCreateManagedWidget(
541         "FEItem_9_0",asciiTextWidgetClass,FEBox[9],
542         XtNeditType,XawtextEdit,
543         NULL);
544     FEItem[0][0] = XtVaCreateManagedWidget(
545         "FEItem_0_0",asciiTextWidgetClass,FEBox[0],
546         XtNeditType,XawtextEdit,
547         NULL);
548     for (i=0;i<8;i++) {
549         sprintf(s,"FEItem_1_%d",i);
550         FEItem[1][i] = XtVaCreateManagedWidget(
551             s,toggleWidgetClass,FEBox[1],
552             XtNresize,False,
553             XtNsensitive,False,
554             XtNlabel,"",
555             NULL);
556     }
557     for (i=0;i<8;i++) {
558         sprintf(s,"FEItem_2_%d",i);
559         FEItem[2][i] = XtVaCreateManagedWidget(
560             s,toggleWidgetClass,FEBox[2],
561             XtNresize,False,
562             XtNsensitive,False,
563             XtNlabel,"",
564             NULL);
565     }
566     for (i=0;i<8;i++) {
567         sprintf(s,"FEItem_3_%d",i);
568         FEItem[3][i] = XtVaCreateManagedWidget(
569             s,toggleWidgetClass,FEBox[3],
570             XtNresize,False,
571             NULL);
572     }
573     for (i=0;i<2;i++) {
574         sprintf(s,"FEItem_4_%d",i);
575         FEItem[4][i] = XtVaCreateManagedWidget(
576             s,commandWidgetClass,FEBox[4],
577             NULL);
578         XtAddCallback(FEItem[4][i],XtNcallback,FEPageCallback,(XtPointer)i);
579     }
580     for (i=0;i<7;i++) {
581         sprintf(s,"FEItem_5_%d",i);
582         FEItem[5][i] = XtVaCreateManagedWidget(
583             s,labelWidgetClass,FEBox[5],
584             XtNjustify,XtJustifyRight,
585             NULL);
586     }
587     for (i=0;i<7;i++) {

```

```

589     sprintf(s,"FEItem_6_%d",i);
590     FEItem[6][i] = XtVaCreateManagedWidget(
591         s,asciiTextWidgetClass,FEBox[6],
592         XtNeditType,XawtextEdit,
593         NULL);
594 }
595 for (i=0;i<7;i++) {
596     sprintf(s,"FEItem_7_%d",i);
597     FEItem[7][i] = XtVaCreateManagedWidget(
598         s,asciiTextWidgetClass,FEBox[7],
599         XtNeditType,XawtextEdit,
600         NULL);
601 }
602 /* FEItem[8][0] = XtVaCreateManagedWidget(
603     "FEItem_8_0",asciiTextWidgetClass,FEBox[8],
604     XtNeditType,XawtextEdit,
605     NULL);
606 */
607 for (i=1;i<6;i++) {
608     sprintf(s,"FEItem_8_%d",i);
609     FEItem[8][i] = XtVaCreateManagedWidget(
610         s,commandWidgetClass,FEBox[8],
611         NULL);
612     XtAddCallback(FEItem[8][i],XtNcallback,
613     FECommandCallback,(XtPointer) i);
614 }
615 */
616 /* _Create the widgets that I need to make the active plot dialog */
617 PMShell = XtVaCreatePopupShell(
618     "PMShell",transientShellWidgetClass,topLevel,
619     XtNtitle,"Plot Manager",
620     NULL);
621 PMForm = XtVaCreateManagedWidget(
622     "PMForm",formWidgetClass,PMShell,
623     NULL);
624 for (i=0;i<4;i++) {
625     sprintf(s,"PMBox_%d",i);
626     PMBox[i] = XtVaCreateManagedWidget(
627         s,boxWidgetClass,PMForm,
628         XtNhSpace,1,
629         XtNvSpace,1,
630         XtNtop,XtChainTop,
631         NULL);
632     sprintf(s,"PMTop_%d",i);
633     PMTop[i] = XtVaCreateManagedWidget(
634         s,labelWidgetClass,PMBox[i],
635         NULL);
636 }
637 XtVaSetValues(PMBox[0],XtNleft,XtChainLeft,NULL);
638 for (i=1;i<4;i++)
639     XtVaSetValues(PMBox[i],XtNfromHoriz,PMBox[i-1],NULL);
640
641 for (i=0;i<8;i++) {
642     sprintf(s,"PMToggle_%d",i);
643     sprintf(t,"Plot %d",i);
644     PMitem[0][i] = XtVaCreateManagedWidget(
645         s,toggleWidgetClass,PMBox[0],
646         XtNsensitive,False,
647         XtNradioData,(caddr_t) i+1,
648         XtNradioGroup,PMitem[0][0],
649         XtNlabel,t,
650         NULL);
651     XtAddCallback(PMitem[0][i],XtNcallback,
652         PMActiveCallback,(XtPointer) i);
653     if (i!=0)
654         XawToggleChangeRadioGroup(PMitem[0][i],PMitem[0][0]);
655 }
656
657

```

```

658     for (i=0;i<8;i++) {
659         PMitem[1][i] = XtVaCreateManagedWidget(
660             "entry", commandWidgetClass, PMBox[1],
661             XtNresize, False,
662             XtNsensitive, False,
663             XtNlabel, "Create",
664             NULL);
665             XtAddCallback(PMitem[1][i], XtNcallback,
666                         PMCreateCallback, (XtPointer) i);
667     }
668     for (i=0;i<8;i++) {
669         PMitem[2][i] = XtVaCreateManagedWidget(
670             "entry", commandWidgetClass, PMBox[2],
671             XtNsensitive, False,
672             XtNresize, False,
673             XtNlabel, "Show",
674             NULL);
675             XtAddCallback(PMitem[2][i], XtNcallback,
676                         PMShowHideCallback, (XtPointer) i);
677     }
678     for (i=0;i<1;i++) {
679         sprintf(s, "PMitem_3_%d", i);
680         PMitem[3][i] = XtVaCreateManagedWidget(
681             s, commandWidgetClass, PMBox[3],
682             XtNresize, False,
683             NULL);
684             XtAddCallback(PMitem[3][i], XtNcallback, PMPlanarCallback, 0);
685     }
686     for (i=1;i<7;i++) {
687         sprintf(s, "PMitem_3_%d", i);
688         PMitem[3][i] = XtVaCreateManagedWidget(
689             s, commandWidgetClass, PMBox[3],
690             XtNresize, False,
691             NULL);
692             XtAddCallback(PMitem[3][i], XtNcallback,
693                         PMActionCallback, (XtPointer) i);
694     }
695
696
697     execShell = XtVaCreatePopupShell(
698         "execShell", transientShellWidgetClass, topLevel,
699         XtNtitle, "Execution Summary",
700         NULL);
701     execForm = XtVaCreateManagedWidget(
702         "execForm", formWidgetClass, execShell,
703         NULL);
704     for (i=0;i<3;i++) {
705         sprintf(s, "execBox_%d", i);
706         execBox[i] = XtVaCreateManagedWidget(
707             s, boxWidgetClass, execForm,
708             XtNhSpace, 1,
709             XtNvSpace, 1,
710             XtNtop, XtChainTop,
711             NULL);
712         sprintf(s, "execTop_%d", i);
713         execTop[i] = XtVaCreateManagedWidget(
714             s, labelWidgetClass, execBox[i],
715             NULL);
716     }
717     for (i=1;i<3;i++)
718         XtVaSetValues(execBox[i], XtNfromHoriz, execBox[i-1], NULL);
719
720     for (i=0;i<4;i++) {
721         sprintf(s, "execItem_0_%d", i);
722         execItem[0][i] = XtVaCreateManagedWidget(
723             s, labelWidgetClass, execBox[0],
724             XtNresize, False,
725             XtNjustify, XtJustifyRight,
726             NULL);

```

```

727     }
728     for (i=0;i<4;i++) {
729         sprintf(s,"execItem_1_%d",i);
730         execItem[1][i] = XtVaCreateManagedWidget(
731             s,asciiTextWidgetClass,execBox[1],
732             XtNeditType,XawtextEdit,
733             NULL);
734     }
735     for (i=0;i<3;i++) {
736         sprintf(s,"execItem_2_%d",i);
737         execItem[2][i] = XtVaCreateManagedWidget(
738             s,commandWidgetClass,execBox[2],
739             XtNresize,False,
740             XtNjustify,XtJustifyCenter,
741             NULL);
742         XtAddCallback(execItem[2][i],XtNcallback,ExecCallback,(XtPointer) i);
743     }
744     fileShell = XtVaCreatePopupShell(
745         "fileShell",transientShellWidgetClass,topLevel,
746         XtNtitle,"Input Data",
747         NULL);
748     filePane = XtVaCreateManagedWidget(
749         "filePane",panedWidgetClass,fileShell,
750         NULL);
751     fileMenuPane = XtVaCreateManagedWidget(
752         "fileMenuPane",panedWidgetClass,filePane,
753         XtNorientation,XtorientHorizontal,
754         XtNshowGrip,False,
755         XtNskipAdjust,True,
756         NULL);
757     fileEntry[0] = XtVaCreateManagedWidget(
758         "fileEntry_0",commandWidgetClass,fileMenuPane,
759         XtNlabel,"Done",
760         XtNshowGrip,False,
761         NULL);
762     fileEntry[1] = XtVaCreateManagedWidget(
763         "fileEntry_1",commandWidgetClass,fileMenuPane,
764         XtNlabel,"Save",
765         XtNshowGrip,False,
766         NULL);
767     fileEntry[2] = XtVaCreateManagedWidget(
768         "fileEntry_2",commandWidgetClass,fileMenuPane,
769         XtNlabel,"Load",
770         XtNshowGrip,False,
771         NULL);
772     for (i=0;i<3;i++)
773         XtAddCallback(fileEntry[i],XtNcallback,FileCallback,(XtPointer) i);
774     fileEntry[3] = XtVaCreateManagedWidget(
775         "fileEntry_3",asciiTextWidgetClass,fileMenuPane,
776         XtNeditType,XawtextEdit,
777         XtNshowGrip,False,
778         NULL);
779     fileText = XtVaCreateManagedWidget(
780         "fileText",asciiTextWidgetClass,filePane,
781         XtNwidth,480,
782         XtNheight,240,
783         XtNeditType,XawtextEdit,
784         XtNscrollVertical,XawtextScrollAlways,
785         XtNshowGrip,False,
786         NULL);
787     /* Create the widgets for the motor geometry dialog */
788     MGShell = XtVaCreatePopupShell(
789         "MGShell",transientShellWidgetClass,topLevel,
790         XtNtitle,"Solid Rocket Motor Geometry",
791         NULL);
792     MGForm = XtVaCreateManagedWidget(
793

```

```

796     "MGForm", formWidgetClass, MGShell,
797     NULL);
798     for (i=0;i<6;i++) {
799         sprintf(s, "MGBBox_%d", i);
800         MGBBox[i] = XtVaCreateManagedWidget(
801             s, boxWidgetClass, MGForm,
802             XtNhSpace, 1,
803             XtNvSpace, 1,
804             NULL);
805         sprintf(s, "MGTOP_%d", i);
806         MGTOP[i] = XtVaCreateManagedWidget(
807             s, labelWidgetClass, MGBBox[i],
808             NULL);
809     }
810     XtVaSetValues(MGBBox[0], XtNleft, XtChainLeft, XtNtop, XtChainTop, NULL);
811     for (i=1;i<3;i++)
812         XtVaSetValues(MGBBox[i], XtNfromHoriz, MGBBox[i-1], NULL);
813     XtVaSetValues(MGBBox[3], XtNfromVert, MGBBox[0], XtNleft, XtChainLeft, NULL);
814     for (i=4;i<6;i++)
815         XtVaSetValues(MGBBox[i], XtNfromHoriz, MGBBox[i],
816             XtNfromVert, MGBBox[0], NULL);
817
818     MGPictShell = XtVaCreatePopupShell(
819         "MGPictShell", transientShellWidgetClass, topLevel,
820         XtNminWidth, 240,
821         XtNminHeight, 240,
822         XtNmaxAspectX, 1,
823         XtNmaxAspectY, 1,
824         XtNsaveUnder, False,
825         XtNtitle, "XXX",
826         NULL);
827     MGPict = XtVaCreateManagedWidget(
828         "MGPict", widgetClass, MGPictShell,
829         XtNwidth, 360,
830         XtNheight, 360,
831         NULL);
832
833     for (i=0;i<8;i++) {
834         sprintf(s, "MGIItem_0_%d", i);
835         MGIItem[0][i] = XtVaCreateManagedWidget(
836             s, labelWidgetClass, MGBBox[0],
837             XtNresize, False, NULL);
838     }
839     for (i=0;i<8;i++) {
840         sprintf(s, "MGIItem_1_%d", i);
841         MGIItem[1][i] = XtVaCreateManagedWidget(
842             s, asciiTextWidgetClass, MGBBox[1],
843             XtNeditType, XawtextEdit, NULL);
844     }
845     for (i=0;i<6;i++) {
846         sprintf(s, "MGIItem_2_%d", i);
847         MGIItem[2][i] = XtVaCreateManagedWidget(
848             s, toggleWidgetClass, MGBBox[2],
849             XtNradioGroup, MGIItem[2][0],
850             XtNradioData, (caddr_t) i+1,
851             XtNresize, False, NULL);
852     }
853     XtVaSetValues(MGIItem[2][0], XtNstate, True, NULL);
854     for (i=0;i<2;i++) {
855         sprintf(s, "MGIItem_3_%d", i);
856         MGIItem[3][i] = XtVaCreateManagedWidget(
857             s, labelWidgetClass, MGBBox[3],
858             XtNresize, False, NULL);
859     }
860     for (i=0;i<2;i++) {
861         sprintf(s, "MGIItem_4_%d", i);
862         MGIItem[4][i] = XtVaCreateManagedWidget(
863             s, asciiTextWidgetClass, MGBBox[4],
864             XtNeditType, XawtextEdit, NULL);

```

```

865      }
866      for (i=0;i<4;i++) {
867          sprintf(s,"MGIItem_5_%d",i);
868          MGIItem[5][i] = XtVaCreateManagedWidget(
869              s,commandWidgetClass,MGBox[5],NULL);
870          XtAddCallback(MGIItem[5][i],XtNcallback,MGCallback,(XtPointer) i);
871      }
872
873  /* Create the widgets for the curve fitting dialog */
874
875  CFShell = XtVaCreatePopupShell(
876      "CFShell", transientShellWidgetClass, topLevel,
877      XtNtitle, "Data Curve Fitting",
878      NULL);
879  CFForm = XtVaCreateManagedWidget(
880      "CFForm", formWidgetClass, CFShell,
881      NULL);
882  for (i=0;i<3;i++) {
883      sprintf(s,"CFBox_%d",i);
884      CFBox[i] = XtVaCreateManagedWidget(
885          s,boxWidgetClass,CFForm,
886          XtNhSpace,1,
887          XtNvSpace,1,
888          XtNtop,XtChainTop,
889          NULL);
890      sprintf(s,"CFTop_%d",i);
891      CFTop[i] = XtVaCreateManagedWidget(
892          s,labelWidgetClass,CFBox[i],
893          NULL);
894  }
895  for (i=1;i<3;i++)
896      XtVaSetValues(CFBox[i],XtNfromHoriz,CFBox[i-1],NULL);
897  for (i=0;i<16;i++) {
898      sprintf(s,"CFItem_0_%d",i);
899      CFItem[0][i] = XtVaCreateManagedWidget(
900          s,toggleWidgetClass,CFBox[0],
901          XtNlabel,"",
902          XtNresize,False,
903          XtNsensitive,False,NULL);
904  }
905  for (i=0;i<16;i++) {
906      sprintf(s,"CFItem_1_%d",i);
907      CFItem[1][i] = XtVaCreateManagedWidget(
908          s,asciiTextWidgetClass,CFBox[1],
909          XtNeditType,XawtextEdit,NULL);
910  }
911  for (i=0;i<4;i++) {
912      sprintf(s,"CFItem_2_%d",i);
913      CFItem[2][i] = XtVaCreateManagedWidget(
914          s,commandWidgetClass,CFBox[2],
915          NULL);
916  }
917
918 }
919

```

